Tratamento de Ambigüidade de
Escopo de Quantificadores
em Processamento de Linguagem Natural

Sandra Maria Aluísio Caldeira

Orientação:

Profa. Dra. Dóris Ferraz de Aragon

*Dissertação apresentada ao Instituto de Ciências Matemáticas de
São Carlos – USP, como parte dos requisitos para obtenção do título
de Mestre em Ciências de Computação e Matemática
Computacional.*

USP - São Carlos
Novembro de 1989

Abstract

Pragmatics is important to handle ambiguity in Natural Language Processing. In particular, pragmatics can be used for quantifier scoping's correct interpretation in natural language sentences.

In this work we treat the problem of handling quantifier scoping in natural language sentences that are assigned a logic meaning representation. In particular, we concentrate in two algorithms that generate quantifier scopings for portuguese sentences.

We address the scoping algorithm where all potencial alternative scoping are generated obtaining the logical form that would correspond to each permutation as well as the scoping algorithm to generate logical forms where exceptions are taken into account to further constraint the number of possibilities.

The two algorithms are presented together with its implementation in Prolog and measures of their efficiency. Although these measures show that the first algorithm is much more efficient than the second algorithm, the problem of efficacy — since after generating the scoping for some sentences, pragmatic information specific to the domain may be essencial for deriving the correct reading — can not be ignored.

Resumo

Neste trabalho, mostramos a importância da pragmática em Sistemas de Processamento de Linguagem Natural. Estamos interessados, principalmente, em utilizá-la na eliminação da ambigüidade inerente à ordem dos quantificadores na representação do significado de sentenças.

Desse modo, analisamos dois algoritmos geradores de escopos — um deles, baseado na geração de todos os escopos combinatoriamente possíveis; e, um segundo que suprime leituras falsas baseando-se na formação das fórmulas lógicas.

Para os dois algoritmos acima, realizamos medidas para análise de eficiência e apresentamos as implementações Prolog dos mesmos.

O problema da eficácia dos algoritmos não pode ser ignorado, desde que o problema de encontrar a representação correta de uma sentença não termina com a geração de escopos alternativos. O conhecimento pragmático deve ser acionado após a geração dos escopos para encontrarmos a interpretação mais plausível para uma dada sentença.

Agradeço

à Profa. Dóris F. de Aragon, pela orientação;

à Profa. Maria Carolina Monard, pelas críticas;

à Graça e à Renata pelo apoio constante;

à Luísa pelo auxílio na produção do texto final; e,

àqueles que de um modo ou de outro contribuíram para este trabalho.

Conteúdo

1	Introdução	1
2	Semântica em Processamento de Linguagem Natural	4
2.1	Alguns Problemas Semânticos	5
2.2	Ambigüidade	9
2.3	Uma Classificação para Teorias Semânticas	11
3	Pragmática em Processamento de Linguagem Natural	18
3.1	Problemas do Âmbito da Pragmática	20
3.2	Modelos de Usuário em Sistemas Cooperativos	22
3.2.1	Modelos de Usuário: O que são e como podem ser usados .	23
3.2.2	Um Modelo de Comportamento Cooperativo	23
3.2.3	Projeto de Sistemas Cooperativos: Problemas e Soluções .	27
4	Determinadores e seus Problemas: Interpretação de Sentenças e Ambigüidade de Escopo	29
4.1	Determinadores	29
4.2	Representação do Significado de Sentenças	31
4.3	Determinadores: Interpretação e Ambigüidade	32

4.4	Tipos de Inferência	33
5	Escopo de Quantificadores: Geração do Melhor Escopo	35
5.1	Tratamento de Determinadores por Dahl e Colmerauer	37
5.1.1	Determinadores e Representação do Significado das Sentenças	37
5.1.2	Escopo das Quantificações	38
5.2	Tratamento de Determinadores no Sistema Chat-80	45
5.2.1	Determinadores e Representação do Significado das Sentenças	45
5.2.2	Escopo das Quantificações	46
5.3	Determinação do Escopo de Quantificadores segundo McCord	50
5.3.1	Modificadores, Estrutura Sintática e Interpretação Semântica	51
5.3.2	Hierarquia Sintática <i>versus</i> Hierarquia Semântica	53
5.3.3	Determinação da Hierarquia Semântica de Modificadores	54
5.4	Considerações de Allen sobre o Escopo de Quantificadores	55
5.4.1	Sentenças com Sintagmas Nominais Simples	56
5.4.2	Sentenças com Sintagmas Preposicionais	58
5.4.3	Sentenças com Cláusulas Relativas	58
5.4.4	Negação	59
6	Escopo de Quantificadores: Geração de Vários Escopos	61
6.1	O Sistema de Reescrita de Saint-Dizier	62
6.1.1	Definição e Construção do Indicador de Domínio	63
6.1.2	Aplicação das Regras de Reescrita sobre o Indicador de Domínio	64
6.1.3	Exemplos de Aplicações das Regras	65
6.1.4	Informação Contextual <i>versus</i> Paráfrases	70

6.2	O Algoritmo de Pereira para a Geração de todas as Permutações de Quantificadores	73
6.3	Eliminação de Leituras Inaceitáveis no Trabalho de Hobbs e Shieber	74
6.4	Tratamento de Quantificadores no Sistema TEAM	77
6.4.1	Escopo de Quantificadores	77
6.4.2	Críticas de Escopo	79
6.4.3	Informações Pragmáticas	79
6.4.4	Interação com o usuário	80
7	Avaliação de Dois Algoritmos Geradores de Escopos	81
7.1	Gerador de Escopos de Pereira	82
7.1.1	Interpretação de Sentenças	82
7.1.2	Linguagem de Entrada e de Saída	83
7.1.3	Descrição do Algoritmo	84
7.2	Gerador de Escopos de Hobbs e Shieber	92
7.2.1	Linguagem de Entrada e de Saída	92
7.2.2	Escopo de Predicados Opacos	94
7.2.3	Descrição do Algoritmo	95
7.3	Tempo de Execução e Avaliação	101
7.4	Considerações: Módulo Pragmático e Módulo de Clarificação de Diálogos	106
8	Conclusão	112
A	Listagens dos Programas Prolog	119
A.1	Interpretação Semântica e Gerador de Escopos de Pereira	119
A.2	Gerador de Escopos de Hobbs e Shieber	127

A.3 Programa para Medir o Tempo de Execucao	134
---	-----

Capítulo 1

Introdução

Imagine que uma agência de viagens de São Paulo possui um Sistema de Processamento de Linguagem Natural (PLN) para acesso a uma Base de Conhecimento, com a finalidade de vender pacotes de viagens para o Rio de Janeiro. Nessa agência, três pessoas — um estudante, um professor de uma universidade e um executivo — utilizam esse sistema dizendo:

Eu gostaria de ir ao Sexto Simpósio em Inteligência Artificial.

O sistema, baseado em conhecimento, prepara os seguintes pacotes de viagens:

- para o estudante: uma passagem de ônibus São Paulo/Rio, dia 06/11; uma reserva por 3 noites no hotel categoria 2 estrelas mais próximo do Hotel Marina Palace; uma passagem de ônibus Rio/São Paulo para o dia 09/11.
- para o professor: uma passagem de avião São Paulo/Rio, dia 06/11; uma reserva por 3 noites no hotel categoria 3 estrelas mais próximo do Hotel Marina Palace; uma passagem de avião Rio/São Paulo para o dia 09/11.
- para o executivo: uma passagem de avião São Paulo/Rio, dia 06/11; uma reserva por 3 noites no hotel categoria 5 estrelas mais próximo do Hotel Marina Palace; uma passagem Rio/São Paulo para o dia 09/11.

O sistema foi bem sucedido porque tem conhecimento de que, para o executivo, a empresa cobre os gastos; para o professor universitário, a universidade auxilia nas despesas; e, o aluno cobre suas próprias despesas. Ainda mais, o sistema sabe

que o simpósio é no Hotel Marina Palace e tem duração de 4 dias, de 6 a 9 de novembro.

Este exemplo mostra que o sistema utilizou conhecimento contextual (conhecimento sobre o simpósio) e conhecimento do mundo (conhecimento sobre professores, executivos, alunos, etc.) para decidir sobre as características do pacote de viagem. Além disso, conseguiu interpretar a frase do usuário como um ato indireto da fala, que, para esse contexto é entendida como uma requisição de um pacote de viagem ao simpósio.

A frase Eu gostaria de ir ao Sexto Simpósio em Inteligência Artificial teria outro significado se fosse enunciada, por exemplo, por um aluno numa sala de aula, diante de um professor que propõe uma prova para o dia 6 de novembro. Em outros contextos poderia ainda ser entendida como uma simples afirmação.

Podemos dizer que o sistema de PLN descrito acima tem embasamento pragmático, pois, para realizar inferências sobre as informações contidas na sua Base de Conhecimento, ele:

- trabalha com modelos de usuário (conhecimento sobre usuários e sobre a sociedade em que vivem);
- utiliza conhecimento contextual; e,
- interpreta sentenças como atos indiretos da fala.

O nosso trabalho analisa esse aspecto da linguagem — sua relação com aqueles que a utilizam. Focalizamos a importância da pragmática para resolver problemas de ambigüidade de escopo de quantificadores. Estes problemas ocorrem quando temos mais de uma interpretação para uma sentença que usa determinadores da linguagem natural como, por exemplo, *um*, *todo*, *muitos* e *cada*.

Apresentamos inicialmente uma classificação para as teorias semânticas aplicáveis a PLN (capítulo 2), utilizando os fundamentos encontrados em [Branco-87] e [Hirst-87]. Abordamos, também, o problema da ambigüidade de sentenças juntamente com outros problemas semânticos como anomalia, contradição e metáforas.

No capítulo 3 mostramos a importância do uso da pragmática discutindo a necessidade de esclarecer as intenções das perguntas, principalmente os fragmentos de sentenças e atos indiretos da fala.

Analisamos no capítulo 4 os problemas relacionados com determinadores — o fato de mudarem a interpretação de uma sentença e o problema da ambigüidade

introduzida pelos quantificadores. Este último problema é chamado ambigüidade de escopo.

No capítulo 5 apresentamos vários trabalhos que utilizam um conjunto de regras heurísticas, definidas *a priori*, para determinar o escopo da quantificação.

Apresentamos no capítulo 6 trabalhos que dão um tratamento mais flexível ao problema de ambigüidade de escopo dos quantificadores. Estes trabalhos estão baseados na geração e comparação de todos os escopos possíveis. A determinação do escopo, neste caso, é obtida através de informações contextuais, interação com o usuário e críticas de escopo.

No capítulo 7 avaliamos dois algoritmos geradores de escopos, baseando-nos em medições do tempo de execução e no número de leituras geradas. Apresentamos também considerações sobre a interação de um algoritmo gerador de escopos com os Módulos Pragmático e para Clarificação de Diálogos.

O capítulo 8 apresenta algumas conclusões e sugestões para trabalhos futuros; o apêndice A contém as listagens dos programas PROLOG que implementam os dois algoritmos geradores de escopos citados acima.

Capítulo 2

Semântica em Processamento de Linguagem Natural

O termo *semântica* figura em textos técnicos e científicos de disciplinas, como Lingüística, Lógica, Psicologia, Filosofia, Linguagens de Programação, Lingüística Computacional e Inteligência Artificial, mas, nem sempre com o mesmo significado.

Uma das razões é que fontes ou motivos não homogêneos levam cada disciplina a formar sua própria teoria para tratar dos seus problemas semânticos específicos.

Segundo Tondl [Tondl-81], não existe, até hoje, uma teoria semântica geral que trate os problemas semânticos dos vários domínios do conhecimento com suficiente generalidade.

O estudo aqui realizado, visando o processamento pela máquina, é relativo a linguagem natural escrita e, em particular, ao Português. Sendo assim, restringimos nossa análise aos sistemas de PLN e, dentro deles, aos sistemas de perguntas e respostas sobre uma base de conhecimento.

Nosso objetivo é utilizar a semântica e a pragmática para compreender o significado das sentenças num sistema de perguntas e respostas.

Um conceito de grande utilidade para a compreensão das partes integrantes de sistemas de PLN é o de interpretação semântica.

Interpretação semântica é o processo de mapear sentenças em linguagem natural (ou sua estrutura sintática) em alguma representação do significado. Esta representação do significado é adequada se mantém constante a intenção original

da pessoa que proferiu a sentença.

Entre as diversas opções para a representação do significado das sentenças em PLN estão vários tipos de lógicas (lógica de primeira ordem, lógicas intencionais, lógica de primeira ordem estendida [Colmerauer-77], [Pereira-83]), primitivas semânticas [Schank-77], *frames* [Hirst-88] ou ainda outras linguagens [Woods-86].

2.1 Alguns Problemas Semânticos

A análise semântica de sentenças de uma linguagem natural deve explicar a maneira como estas sentenças são entendidas, interpretadas e relacionadas com entidades ou ações do universo de discurso. Deve também testar a validade do significado de verbos, adjetivos, preposições e complementos verbais. Isto pode ser conseguido através de uma rede neuronal simples e de uma rede semântica como mostrado em [Branco-87]. Neste trabalho, a rede neuronal expressa as possíveis ligações entre as diversas classes de objetos estabelecidas pelos significados dos verbos, das preposições e de alguns nomes que expressam relações.

A fim de ilustrar a técnica usada por Branco, apresentamos o modo como é interpretada a sentença:

Um homem mora em uma ilha.

Na figura 2.2, a seta valorada com o significado moradia (verbo morar) que liga os nós representativos das classes animal e lugar, diz que o verbo pede um sujeito que seja um animal e um objeto que seja um lugar.

Mas, além desta informação, precisamos saber que a noção de homem está contida dentro da de animal.

Esta informação é obtida pela rede semântica mostrada pela figura 2.1. Ela contém classes e subclasses de nomes próprios e comuns. As setas que ligam dois nós representam o operador *é um*, sendo a classe correspondente ao nó de partida um subconjunto do nó de chegada.

Mas, para obter o significado de uma sentença, deparamo-nos com vários problemas, alguns dos quais são ilustrados pelos seguintes exemplos:

Figura 2.1: Parte da Rede Semântica [Branco-87]

Exemplo 1:

Sua máquina de escrever tem más intenções.

Esta sentença, segundo Bierwisch [Bierwisch-76], é semanticamente anômala, isto é, ela é sintaticamente bem formada, mas carente de significado. Uma sentença semanticamente anômala não possui interpretação, pois as regras semânticas impedem que um sintagma nominal — sua máquina de escrever — possuidor de traço semântico inanimado tenha características que pertençam a objetos animados, neste caso, más intenções.

Há algo mais a ser dito sobre a sentença acima se ela estiver envolvida em um diálogo.

Figura 2.2: Parte da Rede Neuronal Simples [Branco-87]

Para interpretar a sentença *Sua máquina de escrever tem más intenções*, de modo a não se violar o Princípio de Cooperação de Grice [Grice-82] — *em toda comunicação há um acordo geral de cooperação entre um falante e um ouvinte* —, o ouvinte deve supor que o falante está tentando dizer alguma coisa além do significado literal da sentença. Por exemplo, a possibilidade do possuidor da máquina e não a máquina ter más intenções.

Esta sentença pode ser encarada como uma metáfora e, assim, sua interpretação ultrapassa os limites da semântica, devendo ser analisada pela pragmática (Veja o capítulo 3 para maiores detalhes sobre pragmática). Kempson [Kempson-80], pag 77, faz o seguinte comentário sobre o Princípio de Cooperação de Grice:

...parece que o esboço que Grice faz da pragmática proporciona uma base natural para a explicação de como e porque as interpretações metafóricas envolvem a superimposição de uma interpretação (que habitualmente envolve suposições não lingüísticas sobre o mundo) sobre uma outra interpretação, a literal.

Exemplo 2:

Minha irmã solteira é casada com um homem solteiro.

Segundo Bierwisch [Bierwisch-76], esta sentença é contraditória . Isto é, ela é sintaticamente bem formada e possui interpretação, mas, tal interpretação é contraditória pois uma pessoa não pode ao mesmo tempo estar casada e solteira.

Exemplo 3:

João foi até o banco.

Esta sentença é semânticamente ambígua, pois a palavra *banco* pode se referir a um lugar onde se deposita/retira dinheiro; ou, a um lugar para se sentar.

Alguns autores dão outro nome para este tipo de ambigüidade: ambigüidade no sentido da palavra [Carbonell-83] ou ambigüidade léxica [Savadovsky-88].

Em sistemas de PLN, quando o universo de discurso é suficientemente grande para permitir esta ambigüidade, ela pode ser resolvida pelo contexto da sentença

(sentenças anteriores). Entretanto, em um sistema de perguntas e respostas, onde as sentenças são geralmente autônomas, uma solução seria o sistema pedir um esclarecimento ao usuário.

2.2 Ambigüidade

Retirar ambigüidade de sentenças é um dos problemas difíceis em PLN. É tema central deste trabalho o problema de *ambigüidade introduzida pelo escopo de quantificadores*, sendo assim, retornaremos a ela nos próximos capítulos. Um exemplo desse tipo de ambigüidade pode ser visto na sentença:

Um rio corre através de cada país europeu.

Esta sentença possui ambigüidade de escopo de quantificadores, pois, pode ser lida distributivamente, com diferentes rios correndo através de diferentes países europeus; ou, coletivamente com um único rio correndo através de todos os diferentes países europeus.

Este tipo de ambigüidade ocorre quando pode existir mais de um meio de ler a representação do significado da sentença. Casos assim aparecem quando predicados, operadores e descrições incorrem no escopo de negações como *não*; quantificadores como *todo*, *cada*, *muitos*; conjunções e disjunções (*e* e *ou*); e, operadores intencionais (chamados também de opacos) como *acreditar*, *desejar*, *conhecer*, etc.

Além da ambigüidade semântica vista no exemplo 3, temos outros tipos de ambigüidade:

Ambigüidade Sintática

Este tipo de ambigüidade aparece quando temos mais de uma representação sintática para a mesma sentença. Por exemplo, na sentença:

João viu o Corcovado voando para o Rio de Janeiro.

a análise semântica pode resolver a ambigüidade sintática sem requisitar o usuário; pois, as regras semânticas descartarão a interpretação de que um monte voa

— como visto acima, esta interpretação é semanticamente anômala —, restando, assim, somente a interpretação de que João estava viajando de avião para o Rio e viu o Corcovado. Entretanto, se tivéssemos a sentença:

João viu Maria nadando no rio.

o usuário deve ser consultado a fim de fornecer sua interpretação correta se não pudermos obtê-la através de informações contextuais.

A ambigüidade sintática também pode estar presente quando temos diferentes tipos de significado de acordo com a classe da palavra. Por exemplo, a palavra *caça* tem um significado como verbo e outro como substantivo.

Ambigüidade Referencial

Esta ambigüidade é causada por pronomes e outras referências anafóricas. Por exemplo, na sentença:

Coloquei a torta sobre a mesa e a comi.

o ato de comer pode estar relacionado com a torta ou com a mesa. Uma possível solução é armazenar em uma lista todos os sintagmas nominais da sentença e, quando surgir um pronome, procurar na lista a expressão que melhor o substitui, basendo-se no gênero e número associado a ele [Arariboia-88].

Porém, esta solução nem sempre resolve o problema. No caso da sentença acima, onde o gênero e número são os mesmos para os dois sintagmas nominais — a torta, a mesa —, somente o conhecimento do mundo real pode fornecer a interpretação correta.

Ambigüidade de Caso

Um caso refere-se à relação entre um conceito central e um conceito suplementar. Por exemplo, nas sentenças:

Ele correu um quilômetro em quatro minutos.
Ela correu um quilômetro em uma rodovia.

temos uma ambigüidade de caso, pois, a mesma preposição em indica para o ato de correr (conceito central) duas relações diferentes: tempo e localização (conceitos suplementares).

Ambigüidade de Literalidade

A sentença:

Você pode abrir a porta?

pode ser entendida como um pedido; ou, como o questionamento de uma possibilidade física, quando, por exemplo, se discute o lugar onde colocar uma porta na planta de uma casa a ser construída. A ambigüidade reside no fato de se poder interpretar a sentença literalmente (âmbito da semântica) ou como um ato indireto da fala, neste caso uma requisição (âmbito da pragmática).

Para um melhor entendimento do modo de interpretar sentenças como atos da fala, veremos no capítulo 3 um exemplo de um sistema cooperativo que analisa as sentenças de entrada como atos da fala, tentando entender as intenções que levaram o falante a pronunciá-la.

2.3 Uma Classificação para Teorias Semânticas

Segundo Hirst [Hirst-87], um interesse que é comum a muitas teorias semânticas é o princípio da composicionalidade, isto é, quando o significado de toda a sentença é alguma função sistemática do significado de seus componentes. A Semântica de Montague é um exemplo deste princípio.

Hirst cita ainda duas variações em relação ao princípio da composicionalidade. A primeira, quando o significado da sentença é uma função sistemática não somente das partes, mas também do contexto e da situação em que é proferida: a Semântica de Situações.

A segunda variação, quando há em jogo algum tipo de não composicionalidade que pode ou não ser feita através de primitivas semânticas. Por primitivas semânticas entendemos um conjunto finito de termos passíveis de expressar o significado de qualquer palavra e, por implicação, de qualquer sentença. A teoria

da Dependência Conceitual de Schank [Schank-77], por exemplo, usa primitivas semânticas, a Semântica Procedimental não.

A seguir, apresentamos uma classificação para teorias semânticas (figura 2.3) alocando as várias abordagens nos grupos *Lexical*, *Formal* e *Procedimental*, de forma a incorporar as idéias de Hirst [Hirst-87] e a classificação feita por Branco [Branco-87], onde:

A *Semântica de Montague* tem como características a explicitação de condições para a verdade de sentenças e a construção de modelos do mundo; é composicional e usa uma lógica intencional: definição por propriedades ou critérios.

A *Semântica de Situações* tenta formalizar a idéia de uma situação em um mundo real como um possível objeto semântico.

A *Semântica Baseada em Conhecimento* usa uma das formas de se representar o conhecimento — frames, por exemplo — para representar o significado, sendo os elementos da representação considerados como objetos semânticos que fornecem uma interpretação dos constituintes da sentença.

A *Semântica Decomposicional* é tal que o significado de cada palavra é decomposto em um conjunto de primitivas semânticas.

A *Semântica Formal Baseada na Teoria dos Modelos* tenta especificar o significado de cada sentença declarativa com base na Teoria dos Conjuntos e valores de verdade.

A *Semântica Baseada em Valores de Verdade* funda-se nos sistemas lógicos e, praticamente, consiste em considerar o significado da sentença como sua interpretação: atribuição de verdadeiro ou falso no caso da lógica de primeira ordem; e verdadeiro, falso ou indefinido no caso da lógica trivalente. Para maiores detalhes ver no capítulo 5 os trabalhos de Colmerauer e Pereira.

A *Semântica Lógico-Sintática* é tal que as noções de verdade e consequência lógica são reduzidas, respectivamente, à noção de tese (em um sistema formal sintático) e derivabilidade. Para maiores detalhes ver [Branco-87].

A *Semântica Procedimental* usa regras de produção para transformar a sentença de entrada em chamadas de procedimentos que operam sobre uma base de dados.

A seguir, realizamos uma discussão de algumas das teorias semânticas citadas.

As características da *Semântica de Montague* são:

1. o significado de uma sentença é o conjunto das condições necessárias e suficientes para que a sentença seja verdadeira; e,
2. o uso de modelos formais do mundo de modo a relacionar elementos lingüísticos e seus significados. Ela emprega não somente um modelo do mundo, mas, um conjunto de mundos possíveis.

A verdade de uma sentença é relativa a um possível mundo e um ponto no tempo; portanto, a sentença pode ser verdadeira em um mundo e num particular tempo e, falsa em um outro tempo ou em um mundo diferente. Para Montague, objetos semânticos são entidades que podem ser indivíduos no mundo, conceitos sobre indivíduos, propriedades de conceitos sobre indivíduos e funções de ordem mais alta.

Esta teoria usa uma representação intermediária em uma lógica intencional para representar os objetos semânticos, ao invés de traduzir a sentença em linguagem natural diretamente nestes objetos.

O sistema de Montague contém um conjunto de regras sintáticas e um conjunto de regras semânticas onde, cada vez que uma regra sintática é aplicada, há também uma regra semântica para ser aplicada. Devido à essa correspondência de regras e, pelo fato de que a saída de uma regra semântica é sempre uma função sistemática de sua entrada, dizemos que a semântica de Montague é composicional.

Existem dois problemas com respeito a implementação direta da semântica de Montague em sistemas de PLN.

Primeiro porque ela trabalha com conjuntos imensos, funções de funções e pilhas de mundos possíveis, tornando essa implementação direta impraticável computacionalmente.

Um segundo problema é que a semântica de condições de verdade nem sempre é adequada para PLN. Isto porque ela só se interessa pela condição de verdade nos possíveis mundos enquanto que PLN se interessa pelo estado das coisas em si, isto é, por sistemas em que o conteúdo de uma sentença seja retido e representado de uma maneira conveniente.

Para resolver isto, a Semântica de Situações [Barwise-87] dá a sua contribuição. Mas, mesmo assim, é possível usar vários aspectos da teoria de Montague em PLN. Um exemplo é o sistema Absity [Hirst-88].

Barwise e Perry foram os primeiros a apresentar a *Teoria de Semântica de Situações*. Nesta teoria, as sentenças em linguagem natural descrevem situações que têm um tempo associado e um lugar de validade, ou seja, uma região de espaço tempo associada.

Esta teoria se volta aos problemas encontrados na semântica de condições de verdade. Como mencionado anteriormente, esta última nem sempre é adequada para sistemas de PLN pelo fato de que, se a última denotação da sentença é o valor falso ou verdadeiro em um mundo possível, o conteúdo da sentença é perdido.

Além disso, na semântica de condições de verdade todos os comandos logicamente equivalentes têm as mesmas condições de verdade, mesmo se seus conteúdos diferem. Assim, as duas sentenças seguintes, encontradas em [Hirst-87], ambas tautologias, não seriam semânticamente distingüidas, embora digam coisas muito diferentes:

Leste é leste e oeste é oeste.
Ou Nádia chega em tempo ou não.

A *Semântica de Situações* tenta formalizar a idéia de uma situação num mundo real como um possível objeto semântico. Os dois exemplos acima devem se referir a duas situações muito diferentes.

A *Semântica Baseada em Conhecimento* se vale de uma das formas de representar o conhecimento usadas em Inteligência Artificial para manter a representação do significado de uma sentença.

O sistema Absity [Hirst-88] usa *frames* para representar o conhecimento. Nele, nomes e verbos são indentificados com *frames*. A palavra livro, por exemplo, é listada no dicionário do sistema como uma referência a um particular *frame* que descreve livros. Determinadores, como *o* e *um* são seletores de *frames* (funções de recuperação da base de conhecimento) e preposições são *slots*.

Depois que palavras individuais são interpretadas como objetos na representação, elas são combinadas para criar objetos semânticos maiores, que correspondem a constituintes sintáticos no qual estão contidas.

Uma sentença declarativa é interpretada como um comando para adicionar alguma informação — o significado de uma sentença — à uma base de conhecimento. Uma sentença interrogativa é interpretada como um comando que recupera informação de uma base de conhecimento.

As teorias de *Semântica Decomposicional* tentam representar o significado de cada palavra pela sua decomposição em um conjunto de primitivas semânticas. Por exemplo, a palavra *cadeira* pode ser representada da seguinte forma:

(Objeto), (Físico), (Não-Vivo), (Artefato), (Mobília)
(Portátil), (Algo-com-Pernas), (Algo-com-Costas)
(Algo-com-Assento), (Assento-para-Alguém).

Entre as dificuldades encontradas no desenvolvimento das teorias decompositivas está a formação de uma coleção universal de primitivas semânticas, onde todas as palavras (de todas as línguas) pudessem ser decompostas em suas propriedades necessárias.

Outra dificuldade está no modo de representar uma sentença. Isto é problemático, pois, é necessário, de algum modo, combinar as representações léxicas em uma representação da sentença completa e encontrar métodos para inferência no contexto sobre esta representação, de modo a por exemplo, responder a uma pergunta que tal sentença pudesse representar.

Entre os trabalhos sobre Semântica Decomposicional destacam-se as primeiras representações da Teoria de Dependência Conceitual de Schank [Schank-77], o Sistema de Semântica de Preferências de Wilks [Wilks-75], e os Grafos Conceituais de Sowa [Sowa-84].

Abaixo temos exemplos de *Teorias Semânticas Formais Baseadas na Teoria dos Modelos* [Hadley-89].

A *Semântica de Situações* é também considerada como semântica baseada na teoria dos modelos; pois, os significados são fixados por referência a conjuntos de situações abstratas que podem suportar a condição de verdade de uma sentença.

A *Semântica Tarskiana* especifica o significado de uma notação em termos do que suas expressões denotam (isto é, em termos de suas extensões no mundo real). É inadequada se a linguagem em uso inclui predicados intencionais. Na proposição:

acredita(joão, conhece(maria, carlos))

o seu valor verdade não depende do valor verdade de conhece(maria, carlos) — isto seria bem tratado por uma semântica intencional. Para maiores detalhes ver [McDermott-86].

A *Semântica de Mundos Possíveis* especifica o significado de uma sentença em termos da composição de funções definidas extencionalmente, que fornecem mapeamentos de mundos possíveis a extensões e valores de verdade.

Um mundo possível é uma descrição completa do modo que um mundo pode ser (ou pode ter sido). Um desses mundos possíveis é o mundo real e os outros podem ser vistos como variações, começando de mundos onde as mudanças são menores para aqueles onde são grandes as mudanças. A única limitação que um mundo deve ter é ser logicamente consistente.

O termo *Semântica Procedimental* foi criado em 1968 por Woods [Woods-87] para descrever uma metodologia para caracterização do significado de sentenças em PLN, especificamente sistemas de perguntas e respostas. Esta metodologia utiliza procedimentos para caracterizar as condições de verdade de comandos ou de perguntas.

As vantagens da Semântica Procedimental são que as representações lógicas das sentenças, que aqui são usadas como procedimentos, podem ser generalizadas para tratar de um número maior de fenômenos comparados com os formalizados pela lógica de primeira ordem, ou seja:

- procedimentos podem formalizar o significado de ações que causem efeito no mundo; e,
- os mecanismos da lógica formal podem ser estendidos para tratar com sentenças imperativas e interrogativas, além das proposições declarativas.

A classificação para Teorias Semânticas apresentada neste trabalho não é exaustiva nem disjunta. Existem outras classificações, mas, esta pareceu-nos adequada uma vez que analisa duas outras classificações existentes. Das novas abordagens para semântica, a de Situações tem se mostrado muito promissora e merece ser melhor investigada.

Capítulo 3

Pragmática em Processamento de Linguagem Natural

Das três partes do estudo da linguagem — sintaxe, semântica e pragmática — esta última é a menos sistematizada e cujos estudos formais ainda não receberam atenção especial de lógicos, filósofos e lingüistas. São vários os autores que destacam este ponto, entre eles podemos citar Clifford [Clifford-88]:

A pragmática é o ramo menos compreendido da divisão tripartida do estudo da linguagem que Morris propôs em sua teoria de semiótica. Este século presenciou um sucesso tremendo no desenvolvimento da sintaxe lógica formal e da semântica baseada na teoria dos modelos, mas viu muito pouco do desenvolvimento de uma pragmática formal.

e Dascal [Dascal-82]:

Porque, tendo sido durante muito tempo encarada como a lata de lixo da lingüística, a pragmática, apesar de ser hoje séria e intensamente investigada, ainda parece mais um amontoado heterogêneo de dados, problemas e pontos de vista do que um domínio teórico bem definido.

Portanto, é função dos filósofos da linguagem delinear os problemas fundamentais da pragmática e sua área de atuação. Aos lógicos e lingüistas cabe realizar uma caracterização adequada de seus conceitos e uma análise de sua utilização. Aos pesquisadores da área de Lingüística Computacional cabe, enfim, avaliar os

conceitos teóricos e os avanços das áreas de Lingüística, Lógica e Filosofia para uma possível aplicação em sistemas de PLN.

Por outro lado, como o estudo das linguagens naturais em suas diversas facetas é elaborado dentro da própria linguagem, muitos de seus aspectos permanecem confusos.

Nos sistemas formais, uma caracterização precisa se faz entre a linguagem objeto e a metalinguagem permitindo o esclarecimento de dúvidas e a precisão de noções que, dentro da própria linguagem, poderiam levar a circularidades.

Em Lingüística, o estudo da linguagem se faz com o uso da própria linguagem, o que é inconveniente. Uma metalinguagem para a análise da linguagem deve atender a pelo menos três requisitos:

- todas as expressões da linguagem objeto devem ser expressas nessa metalinguagem;
- a metalinguagem deve conter conceitos semânticos em relação a linguagem objeto; e,
- a análise dos aspectos pragmáticos da linguagem deve ser passível de elaboração nesta metalinguagem.

Existem autores que consideram a pragmática como parte da semântica. Allen [Allen-87], por exemplo, não faz uma delimitação clara do âmbito da semântica e da pragmática. Em nosso trabalho, entretanto, vamos distinguir a pragmática da sintaxe e da semântica.

O conhecimento pragmático permite através de informações contextuais determinar o significado adequado para as sentenças. Já, o conhecimento do mundo é necessário para que seja possível em muitos casos retirar a ambigüidade e permitir um melhor entendimento das sentenças onde o diálogo se realiza.

Veremos agora como os autores [Stalnaker-82] e [Arariboia-88] abordam a pragmática.

Segundo Stalnaker a pragmática é o estudo dos atos lingüísticos e dos contextos nos quais eles são executados. Por atos lingüísticos ou atos da fala entende-se: *afirmações, ordens, respostas, informações, alegações, pedidos, predições, promessas, suposições, insultos, etc.*

Segundo Arariboia, a força e a riqueza da linguagem natural reside nos significados implícitos. O tipo de significado implícito que, para ser detectado, é

preciso saber quais as intenções do falante e qual o contexto em que uma frase foi proferida está relacionado com a pragmática.

Ainda, para Arariboia, a pragmática tem seu papel fundamental quando trabalhamos com um conjunto de frases — em um discurso ou em um diálogo —; pois, para frases isoladas de um contexto é possível que os dados da semântica sejam suficientes para interpretá-las. Com um conjunto de frases, a descoberta da intenção embutida nas sentenças ajudará a completar o significado da sentença. Como conclusão, temos que é trabalho da pragmática examinar as relações entre os usuários da linguagem e o contexto de uso das sentenças.

A seguir citamos alguns dos problemas que certamente devem ser tratados dentro da pragmática.

3.1 Problemas do Âmbito da Pragmática

Já mencionamos que a pragmática é a menos estudada das três partes que compõem a semiótica, e, cujo progresso em PLN tem sido lento.

Assim como a semântica só ganhou impulso com um tratamento preciso de noções como denotação, verdade, acarretamento, etc., pensamos haver uma necessidade imperiosa de considerarmos uma pragmática formal. Porém, as dificuldades da precisão e unificação serão bem maiores uma vez que a pragmática deve levar em consideração o usuário, tornando-se assim muito mais subjetiva.

Como poderemos assim precisá-la já que o grau de subjetividade está incluído?

No caso da semântica, consideramos universos como modelos objetivos de interpretação, no caso da pragmática temos que considerar aspectos que direcionem os universos considerados através dos modelos do falante/usuário.

É muito importante considerarmos, por essas razões, as crenças, os objetivos, os planos, o conhecimento do usuário e do contexto onde ele se insere.

Existem dois problemas principais que devem ser resolvidos pela pragmática.

O primeiro problema é definir tipos relevantes de atos da fala. O segundo é caracterizar os traços do contexto de fala que ajudam a determinar o que é expresso por uma sentença.

Um estudo aprofundado do primeiro tipo de problema pode ser visto em [Allen-80] e [Perrault-86].

Para o tratamento da ambigüidade de escopos é importante estudar o segundo tipo de problema, no qual se inserem as pressuposições.

O significado de um texto não é somente a soma dos significados das sentenças que ele contém. Cada sentença contém um significado oculto. Podemos dizer que as pressuposições envolvem fatos que um falante não enuncia por supor que o ouvinte já conheça. Vamos dar alguns exemplos que envolvem determinadores:

O primo do presidente será o ministro da agricultura.

Fato pressuposto: O presidente tem um primo.

Certos candidatos são democratas.

Fato pressuposto: Há candidatos não democratas.

Os guerreiros de Ulisses não viveram.

Fato pressuposto: Ulisses teve guerreiros.

Telêmaco conhece a ambição de cada pretendente de Penélope.

Fato pressuposto: Penélope tem pretendentes.

Esses problemas acima abordados dizem respeito a sentenças declarativas; mas, em sistemas de PLN, temos que levar também em consideração sentenças interrogativas e sentenças imperativas. As noções que correspondem ao valor de verdade das sentenças declarativas são nas sentenças interrogativas a coerência de respostas e, nas sentenças imperativas o desempenho da instrução.

Sabemos que a ambigüidade ocorre em todos os níveis de análise da sentença, em muitos casos, sem o conhecimento do contexto onde a sentença está sendo enunciada não se pode identificar a leitura apropriada. Desta forma, neste trabalho é destacado o nível pragmático para obtermos a leitura correta em sentenças com ambigüidade de escopo de quantificadores.

Em sistemas de perguntas e respostas, de forma explícita ou não, um modelo dos participantes dos processos conversacionais deve ser levado em consideração. Isso é bem ressaltado no capítulo sobre PLN de Arariboia onde é dito que *para os diálogos de um sistema de perguntas e respostas é preciso levar em consideração os usuários como interlocutores e colocar como uma questão pertinente da pragmática a necessidade de modelos de usuário.*

3.2 Modelos de Usuário em Sistemas Cooperativos

Uma das características da comunicação entre pessoas é que elas adquirem e usam um conhecimento considerável sobre as pessoas com as quais conversam. Por exemplo, se desejamos que a interação de pessoas com máquinas em um sistema de perguntas e respostas que acesse uma base de conhecimento seja confortável, isto é, se processe de uma maneira natural, devemos permitir que o sistema adquira e use conhecimento das pessoas com quem interage.

De acordo com Grice [Grice-82], o uso da linguagem natural para a comunicação inclui um conjunto de convenções que devem ser seguidas em um diálogo. Por exemplo, um ouvinte deve acreditar que o falante acredita que ele (ouvinte) pode inferir o significado de uma fala, caso contrário o falante não a diria. Sendo assim, dada uma sentença mal formada pragmaticamente, um ouvinte cooperativo deveria tentar deduzir as intenções do falante.

Uma pessoa que interage com um computador via linguagem natural irá assumir que essas convenções estão sendo seguidas e ficará muito insatisfeita se elas não o estiverem. Estas convenções requerem de um sistema que permita a comunicação em linguagem natural o conhecimento sobre os objetivos, planos, capacidades, atitudes, e crenças das pessoas com quem interage.

Teremos, assim, sistemas interativos inteligentes, chamados resolvidores de problemas cooperativos. Exemplos de tais sistemas são as interfaces inteligentes para sistemas especialistas e base de dados, sistemas tutoriais inteligentes e sistemas de ajuda e consulta.

Veremos a seguir uma caracterização de modelos de usuário e como podem ser usados.

3.2.1 Modelos de Usuário: O que são e como podem ser usados

O termo *modelo de usuário* é usado para descrever o conhecimento empregado para dar suporte a uma interface homem-máquina.

A definição dada por Kass e Finin [Kass-88] é a seguinte:

Um modelo de usuário é o conhecimento sobre o usuário, explicitamente ou implicitamente codificado, que é usado pelo sistema para melhorar a interação.

O conhecimento sobre um usuário que é fornecido pelo modelo pode ser usado de vários modos em um sistema de PLN. Os vários modos são geralmente categorizados em uma taxionomia que é mostrada na figura 3.1.

3.2.2 Um Modelo de Comportamento Cooperativo

Allen e Perrault [Allen-80] descrevem um modelo de comportamento cooperativo e como ele pode ser aplicado a sistemas de PLN. Eles assumem que agentes tentam reconhecer os planos de outros agentes e, então, usam esse plano para fornecer respostas.

No estudo de como o modelo proposto pode ser usado para explicar vários aspectos do comportamento lingüístico, Allen e Perrault partem de três suposições:

- pessoas são agentes racionais que são capazes de fazer e executar planos para alcançar seus objetivos;
- elas são frequentemente capazes de inferir os planos de outros agentes pela observação do desempenho da ação daquele agente; e,
- elas são capazes de detectar *obstáculos* no plano de outros agentes; onde obstáculos são objetivos em um plano que o outro agente não pode alcançar (facilmente) sem assistência.

Uma forma de comportamento prestativo aparece devido ao fato de um agente ter um objetivo de superar um obstáculo no plano de outro agente. Segundo Allen e Perrault, a linguagem pode ser vista como uma instância desse comportamento orientado a objetivos:

Figura 3.1: Usos de Modelos de Usuário [Kass-88]

A fala é produzida por ações (atos lingüísticos) que são executados de modo a ter algum efeito sobre o ouvinte. Este efeito envolve tipicamente a modificação das crenças ou objetivos do ouvinte. Um ato da fala, como outra ação qualquer, pode ser percebido pelo ouvinte e permitir que ele infira qual é o plano do falante.

Como exemplo de uma aplicação para PLN podemos ter um dos agentes sendo um sistema de perguntas e respostas que tira dúvidas ou fornece informações sobre um assunto que está em sua base de conhecimento. O outro agente seria um usuário do sistema — aquele que possui as dúvidas ou busca informações.

Frequentemente, um ato da fala pode explicitamente comunicar ao ouvinte um objetivo que é um obstáculo ao falante; mas, podem existir outros objetivos no plano que não foram explicitamente comunicados, e que, também são obstáculos. Uma resposta útil tentará superar os obstáculos explícitos e implícitos.

Por exemplo, considere a seguinte troca de informação que ocorre em um guichê de trem:

Passageiro: Quando parte o trem para Rio Claro?

Atendente: 3:15, na plataforma 7.

A pergunta do passageiro exprime que ele deseja saber o horário de partida do trem. Mas como pode ser observado pela resposta do atendente, havia um obstáculo implícito — o passageiro precisa saber a plataforma de embarque — que o atendente soube reconhecer e superar.

Portanto, um ouvinte prestativo não deve basear sua resposta somente nos obstáculos explícitos, deve procurar também os implícitos. Outro exemplo seria:

Um agente A está carregando um galão de gasolina vazio e ao encontrar o agente S em uma rua, pergunta:

Onde fica o posto de gasolina mais próximo?

S responde:

Na próxima esquina.

mesmo sabendo que o posto está fechado. O agente S, portanto, não foi de grande ajuda. Um agente cooperativo notificaria à A outros obstáculos existentes em seu plano, principalmente aqueles de que A não está ciente. Este é o comportamento esperado, mesmo quando A e S são estranhos. Se A acredita que S sabe de antemão que o posto estava fechado, ele tem toda razão para ficar desapontado com S, pois ele violou as regras básicas da cooperação humana.

No modelo cooperativo, Allen e Perrault estão interessados nos atos de requisição, advertência, declarações (afirmações) e promessas. Esses atos podem requisitar dos interlocutores certas crenças e intenções (desejos). Por exemplo, para *informar* sinceramente a alguém que eu estou cansada, duas condições devem ocorrer (precondições para o ato de informar):

1. eu tenho que acreditar que eu estou cansada e
2. devo pretender fazer com que a pessoa para a qual vou informar acredite que eu estou cansada.

Assim, o efeito desse ato será o de que, agora, a pessoa para a qual dirigi meu ato acredita que eu estou cansada.

Este modelo fornece mecanismos para esclarecer alguns aspectos interessantes da linguagem:

- a geração de respostas que fornecem mais informação que a requerida.
- a geração de respostas para fragmentos de sentenças e atos indiretos da fala.
- a geração de subdiálogos para esclarecer a intenção de uma pergunta anterior.

Uma descrição intuitiva do que ocorre quando um agente A faz uma pergunta a outro agente B, que então responde, seria:

A tem um *objetivo* de adquirir alguma informação; ele cria um plano (*construção do plano*) que envolve fazer uma pergunta a B cuja resposta fornecerá a informação. A então executa seu plano, fazendo a pergunta a B. O agente B recebe a pergunta e tenta inferir o plano de A (*inferência do plano*). Neste plano pode existir objetivos que A não pode realizar sem assistência. Esses são os *obstáculos* no plano de A.

B pode admitir alguns desses obstáculos como seus próprios objetivos e criar um plano para superá-los. A resposta de B é gerada quando ele executa seu plano.

Como conclusão, o funcionamento do modelo do sistema cooperativo pode ser resumido assim:

Se o contexto é restrito suficientemente para determinar o plano do falante, então, respostas apropriadas são geradas para uma grande variedade de sentenças consideradas problemáticas. Se o contexto não é suficientemente restrito, o modelo gera sentenças que levam a um diálogo para esclarecimentos.

3.2.3 Projeto de Sistemas Cooperativos: Problemas e Soluções

Os estudos realizados por Cohen [Cohen-85] estão baseados em interfaces em linguagem natural para acesso a banco de dados e sistemas especialistas. Estes estudos demonstram a necessidade de uma análise pragmática e a criação de modelos de usuário para garantir respostas efetivas por parte do sistema.

Mostramos abaixo vários problemas encontrados no projeto de interfaces para sistemas interativos e algumas soluções para eles.

Divergência entre a Visão do Usuário e do Sistema sobre o Mundo.

O problema é que perguntas de usuários frequentemente escondem aspectos de seus objetivos, crenças e intenções que deveriam influenciar a resposta do sistema. Falhas do sistema na correção de pressuposições deverá propagar enganos, isto é, se o sistema e o usuário possuem crenças diferentes sobre determinado fato, o sistema deveria corrigir a falsa pressuposição. Para que o sistema possa detectar as crenças do usuário, ele deveria projetar um modelo do conhecimento do usuário para detectar prováveis inferências.

Problemas de Referência, Ambigüidade Sintática e no Sentido das Palavras.

Para que o sistema não falhe em detectar referências nem forneça respostas erradas em frases ambíguas, ele deveria reter a representação da pergunta do usuário; deveria traçar o foco de atenção e os objetivos para responder corretamente as perguntas subsequentes. Isto é levado em consideração no sistema de Branco [Branco-87], que consegue responder as seguintes perguntas:

- Onde mora Odisseus ?
- Em Ítaca.
- E Penélope ?
- Em Ítaca.

Como fornecer a Melhor Resposta para um Diálogo que Ajuda na Busca de Informações.

A ênfase neste caso é na construção de um plano de intenções do usuário a medida que o diálogo progride. A estratégia é manter uma hierarquia de planos onde o objetivo de nível mais baixo é inferido do diálogo e relacionado com os planos de níveis superiores que tratam do contexto global.

Problemas nas Referências à Objetos no Universo de Discurso.

De modo a entender a referência de um usuário, um sistema deveria ser capaz de consultar seu próprio modelo de mundo e relaxar parte das descrições que não se igualam. Para isso é necessário que o sistema mantenha um modelo de suas próprias crenças para comparação, como um meio de entender perguntas que de outra forma seriam ininteligíveis.

Capítulo 4

Determinadores e seus Problemas: Interpretação de Sentenças e Ambigüidade de Escopo

4.1 Determinadores

Certos sintagmas nominais são compostos por *palavras* que determinam, por exemplo, quantos objetos estão sendo descritos ou como os objetos se referem ao falante e ouvinte. Estas palavras, como mostrado abaixo, não pertencem a uma classe sintática única na gramática da língua portuguesa, por exemplo:

cada casa, toda casa, cuja casa, minha casa, várias casas, a casa, nenhuma casa, duas casas, poucas casas, esta casa, qual casa, uma casa, etc.

Na gramática inglesa, elas são geralmente chamadas de *determiners* [McCord-87] e também são, juntamente com adjetivos e outros substantivos, os principais tipos de premodificadores de substantivos.

Esta descrição, que não é usual na gramática portuguesa, encontra-se em [Melo-78] com o nome de determinantes e em [Luft-85] também com o nome de determinativos — nome oriundo de pronomes adjetivos determinativos. Neste trabalho chamaremos estas palavras de *determinadores*.

Em Linguística Computacional não existe um consenso na consideração dos determinadores. McCord, a eles se refere como a classe que determina:

- a quantos objetos nos referimos (cada, todo, algum, muitos, poucos, nenhum, ambos, etc.);
- como o objeto que está relacionado ao falante (este (aqui), esse (ai), aquele (lá), etc.);
- se o objeto é ou não unicamente indentificável (o *versus* um);
- se o objeto necessita ser identificado pelo ouvinte (que, cujo, qual, etc.);
- o número de objetos que estão sendo descritos (um, dois, três, etc.);
- se há posseção em relação ao objeto descrito (meu, seu, nosso, dele, etc.); e,
- a ordem de um objeto em um conjunto (primeiro, segundo, etc.).

Colmerauer [Colmerauer-77], no entanto, considera apenas alguns determinadores da classe acima, denominando-os artigos: um, todo, cada, o, a, muitos, poucos, nenhum, todos os, etc. Para ele não são determinadores as seguintes situações:

- como o objeto está relacionado ao falante;
- se o objeto necessita ser identificado pelo ouvinte;
- se há posseção em relação ao objeto descrito; e,
- a ordem de um objeto em um conjunto.

Saint-Dizier [Saint-Dizier-85] trata os determinadores colocando-os em quatro classes:

$D = \{\text{todo, cada}\}$

$E = \{\text{um(a), algum(a), alguém, muitos, ninguém, muitos dos, poucos, quem, qual, a(o) qual, o, ...}\}$

$Q = \{\text{uma porção, um pouco, ...}\}$

$U = \{\text{os, todos os, todos, nenhum, ...}\}$

Saint-Dizier não trata, portanto:

- como o objeto está relacionado ao falante;
- se há posse em relação ao objeto descrito; e,
- a ordem de um objeto em um conjunto.

Neste trabalho usaremos os artigos de Colmerauer como determinadores. Eles são chamados *quantificadores* na representação do significado das sentenças. Aqui, chamaremos de operadores os *quantificadores*, os jutores (por exemplo: não, e, ou, implicação) e também os qualificadores (τ e λ).

4.2 Representação do Significado de Sentenças

As limitações do cálculo de predicados — lógica de primeira ordem —, para a representação de sentenças, estão relacionadas principalmente com quantificações. O cálculo de predicados fornece somente duas escolhas: universal e existencial; e, as variáveis que a elas podem ser aplicadas são restritas a representar elementos do domínio, excluindo funções e predicados. Também os valores de verdade fornecidos pelo cálculo de predicados são verdadeiro e falso, não deixando margem para a incerteza.

A linguagem natural está permeada de noções quantitativas de vários tipos:

- número de objetos (duas bolas, três maçãs, etc.);
- quantidade de substâncias (muito açúcar, pouco mel, etc.);
- graus de intensidade (muita força, pouco sofrimento, etc.); e,
- ordenação na referência a objetos ou a tempo (terceira rua, quinto dia, etc.), etc.

O suporte para o conceito de quantidade não é bem fornecido pela lógica de primeira ordem pura. Todavia, é possível construir representações para esses conceitos em modelos lógicos.

São duas as principais posições quanto a consideração de lógica de primeira ordem enquanto instrumento para a representação do significado de sentenças.

A primeira procura estender esta lógica considerando novos operadores. Como em Pereira [Pereira-83] que cria dois operadores — *setof* e $\setminus +$ (não provabilidade)

— e, com isso, se afasta das propriedades fundamentais da lógica de primeira ordem (como a monotonicidade). Em trabalhos mais recentes, Pereira abandona esta abordagem e passa a usar o cálculo λ para o tratamento de quantificadores [Pereira-86]. Já, Colmerauer prefere uma lógica trivalente e estende as noções semânticas da lógica de primeira ordem a fim de englobar situações [Colmerauer-77].

A segunda posição é a do uso de outras lógicas. Como Montague que usa uma lógica intencional e outros autores que usam tanto a lógica default [Saint-Dizier-88] como lógicas nebulosas para o tratamento de quantificadores.

4.3 Determinadores: Interpretação e Ambigüidade

Um problema relacionado com o uso de determinadores é o fato de modificarem e complementarem a interpretação de uma sentença. Considere as três sentenças encontradas em Savadovsky [Savadovsky-88]:

Todo engenheiro gosta de coisas complexas.
Algum engenheiro gosta de coisas complexas.
O engenheiro gosta de coisas complexas.

Embora o sintagma verbal seja o mesmo nas três sentenças, na primeira, o determinador **todo** introduz uma quantificação universal sobre o conjunto de engenheiros. Na segunda, o determinador **algum** introduz uma quantificação existencial; enquanto que, na terceira, uma interpretação possível seria a de pressupor a existência e unicidade de um particular **engenheiro**.

Outro problema é a determinação do escopo da quantificação que é introduzida pelos determinadores. A sentença:

Toda criança possui um brinquedo.

é ambígua; pois, tanto a quantificação do determinador **toda**, quanto a quantificação do determinador **um** podem ter o maior escopo, isto é, ser a mais externa.

As duas representações em lógica de primeira ordem são, respectivamente:

- 1) $(\forall C)crianca(C) \Rightarrow ((\exists B)(brinquedo(B) \& possui(C, B))$
- 2) $(\exists B)brinquedo(B) \& ((\forall C)crianca(C) \Rightarrow possui(C, B))$

A interpretação da representação 1) é a de que existem diferentes brinquedos, cada um pertencente a uma criança e, para a da representação 2), é a de que existe um brinquedo, por exemplo, *um ursinho de pelúcia*, que pertence a todas as crianças.

Vários trabalhos relacionados com o escopo da quantificação introduzida pelos determinadores serão discutidos nos capítulos 5 e 6. Embora possa se fazer uma escolha *a priori* quando duas ou mais configurações são possíveis, consideramos que a hipótese mais adequada é aquela que faz uso de informação contextual (fatos na base de dados, relações sobre estes fatos, testes de integridade) para selecionar a interpretação correta de uma sentença. Nosso posicionamento é análogo a Woods [Woods-86] que diz:

Em geral, a maneira mais correta de manusear muitas das ambigüidades potenciais no Inglês é construir representações de interpretações alternativas; ou, partes de interpretações alternativas, avaliando as alternativas de acordo com sua aceitabilidade.

Na obtenção de informações contextuais, muitas vezes realizamos inferências para tirar a ambigüidade da linguagem natural. Faremos a seguir um breve resumo dos tipos de inferências para PLN.

4.4 Tipos de Inferência

Para o processamento de linguagem natural são necessárias muitas formas diferentes de inferência. Vamos agrupá-las em inferências dedutivas e não dedutivas.

As formas dedutivas são baseadas na noção lógica de acarretamento. Isto é, dado um conjunto de fatos, um processo de inferência dedutivo levará somente à conclusões que logicamente seguem daqueles fatos. Se a representação do conhecimento usa somente inferência dedutiva pura, dizemos que é uma representação monotônica; e, se novos fatos consistentes são adicionados à base de conhecimentos, tudo que previamente foi inferido continua sendo verdadeiro. A adição de fatos novos somente aumenta o número de proposições que são válidas.

Como exemplos de inferência não dedutiva temos a inferência envolvendo o aprendizado de generalizações de exemplos — inferência indutiva — e a inferência envolvendo a busca da interpretação mais plausível para uma sentença sendo dado seu contexto. Neste caso, algumas vezes podemos usar também inferências dedutivas.

Outra classe de inferência está relacionada ao uso de informações *default*, isto é, uma regra *default* é uma regra de inferência para a qual pode existir exceções. Por exemplo, a sentença Pássaros voam pode ser representada como:

$$(\forall X) \text{passaro}(X) \Rightarrow \text{voa}(X)$$

significando que todos os pássaros voam, a menos que explicitamente se diga o contrário.

Todo sistema com regras *default* é não-monotônico. Se temos os fatos:

$$\begin{aligned} &(\forall X) \text{passaro}(X) \Rightarrow \text{voa}(X) \\ &(\forall X) \text{pinguin}(X) \Rightarrow \text{nao}(\text{voa}(X)) \\ &\text{passaro}(\text{Piupiu}) \end{aligned}$$

podemos concluir que Piupiu voa. Mas se Piupiu for pinguim:

$$\text{pinguin}(\text{Piupiu})$$

o sistema deverá concluir que Piupiu não voa. Desde que Piupiu seja pássaro e pinguim, o sistema deve priorizar regras com informações específicas que permitam a conclusão do não vôo, mesmo para um pássaro.

Capítulo 5

Escopo de Quantificadores: Geração do Melhor Escopo

No presente trabalho estamos interessados na eliminação da ambigüidade introduzida por problemas de escopo de quantificadores.

Existem muitos trabalhos relacionados com o escopo de quantificadores que apresentam apenas um conjunto de regras heurísticas, definidas *a priori*, para determinar o escopo da quantificação [Colmerauer-77], [Dahl-81], [Pereira-82], [McCord-82], [Pereira-83], [Woods-86] e [Allen-87].

As regras mais comuns são as seguintes:

- heurísticas léxicas: por exemplo, *cada* tem escopo maior que *algum*;
- heurísticas sintáticas: por exemplo, o quantificador de um sintagma nominal em uma cláusula relativa é governado pelo quantificador do sintagma nominal principal; e,
- heurísticas de disposição: a ordem da esquerda para direita dos determinadores na sentença é geralmente tomada como *default*.

Embora tais regras produzam interpretações corretas da sentença em linguagem natural, em alguns casos esse conjunto fixo de regras pode levar a falsas interpretações; pois, a interpretação correta irá depender do contexto e/ou do conhecimento sobre o mundo.

Mostraremos a seguir dois exemplos de [Saint-Dizier-85], onde uma escolha *a priori* do escopo de quantificadores levaria a interpretações erradas:

Todos os estudantes trabalham com um microcomputador.

Todos os estudantes trabalham com um Vax.

A interpretação correta da primeira sentença ocorre quando a quantificação introduzida pelo determinador todos os domina a quantificação introduzida pelo determinador um. Entretanto, a interpretação correta da segunda sentença ocorre quando a quantificação introduzida pelo determinador um domina a quantificação introduzida pelo determinador todos os. Isto decorre dos seguintes fatos:

Um Vax é freqüentemente um computador compartilhado.

Um micro é um computador pessoal.

O uso destas informações sobre computadores, que recaem sobre o âmbito da pragmática, ajudaria a encontrar a interpretação correta para as duas sentenças.

Woods [Woods-86], tece um comentário a esse respeito:

Na divisão clássica de problemas de significado nas áreas da sintaxe, semântica e pragmática, a última é usada para denotar aqueles aspectos do significado determinados não por regras semânticas comuns, mas por aspectos da situação corrente, conhecimentos do falante, etc.

São regras semânticas comuns as redes semânticas e as neuronal simples do trabalho de [Branco-87]. Exemplos de conhecimento do falante são, por exemplo, conhecimentos sobre fatos do mundo.

Temos abaixo um resumo de vários trabalhos relevantes que enfocam bem o problema do escopo da quantificação introduzida pelos determinadores da linguagem natural.

Procuramos não só adaptar para o Português a terminologia usada nesses trabalhos, mas também precisar as noções que neles se encontram vagas.

5.1 Tratamento de Determinadores por Dahl e Colmerauer

5.1.1 Determinadores e Representação do Significado das Sentenças

Colmerauer [Colmerauer-77] e Dahl [Dahl-81] representam as sentenças por uma árvore quantificada com três ramos:

$$q(x, f1, f2)$$

sendo q um quantificador onde o determinador é traduzido; $f1$ a tradução do sintagma nominal principal e $f2$ a tradução do sintagma verbal.

Vejamos a representação da sentença O rei corajoso ganhou um navio:

Podemos dizer que, na árvore quantificada de Colmerauer e Dahl:

- a rotulação dos nós se dá por quantificadores introduzidos por determinadores, por conjunções que representam sentenças mais complexas ou por negações;

- as folhas são predicados, variáveis ou constantes.

Dahl comenta que é útil representar todas as quantificações através de uma única forma, ao invés de gerar quantificadores diferentes para cada determinador.

Portanto, a cada quantificação é atribuída uma expressão *para* equivalente (estrutura lógica), na qual é representada o significado dos determinadores usados no sistema lógico:

$um(x, f1, f2) = para(x, e(f1, f2), maior_que(card(x), 0))$
 representa também os determinadores *algum* e *alguém*
 $todo(x, f1, f2) = para(x, e(f1, não(f2)), igual(card(x), 0))$
 representa também os determinadores *cada* e *qualquer*
 $o(x, f1, f2) = para(x, f1, se(igual(card(x), 1), f2))$
 $os(x, f1, f2) = para(x, f1, se(maior_que(card(x), 1), f2))$
 $nenhum(x, f1, f2) = para(x, e(f1, f2), igual(card(x), 0))$
 representa também o determinador *ninguém*
 $i(x, f1, f2) = para(x, e(f1, f2), igual(card(x), i))$
 representa qualquer número cardinal
 $alguns(x, f1, f2) = para(x, e(f1, f2), maior_que(card(x), 1))$
 representa também o determinador *uns*
 $nem_todo(x, f1, f2) = para(x, e(f1, não(f2)), maior_que(card(x), 0))$

As fórmulas do tipo *card* indicam a cardinalidade esperada do conjunto res-posta. Para maiores esclarecimentos confira [Assis-86].

5.1.2 Escopo das Quantificações

As heurísticas usadas por Dahl e Colmerauer para a determinação do escopo *correto* dos quantificadores são de dois tipos: Heurísticas Sintáticas e Léxicas.

Hierarquia dos Quantificadores

A hierarquia dos quantificadores obedece a três regras:

Regra 1: A quantificação introduzida pelo determinador do sujeito domina as quantificações introduzidas pelos complementos do verbo.

A sentença Todos os homens têm uma lança é representada por:

Uma lança é tida por todos os homens, que é a sentença anterior na voz passiva, é representada por:

A mudança da voz ativa para a passiva inverte a hierarquia dos quantificadores, isto é, fornece uma distinção entre as duas vozes.

Regra 2: Em uma construção envolvendo um nome e um complemento, a quantificação introduzida pelo determinador do complemento domina a quantificação introduzida pelo determinador do nome.

Em Telêmaco conhece a ambição de cada pretendente temos:

Regra 3: Quando um verbo, um adjetivo ou um nome tem dois complementos, a quantificação é feita da direita para a esquerda: o complemento mais a direita gera a quantificação que domina a quantificação gerada pelos complementos a esquerda.

Para Ulisses dá um copo de vinho a cada guerreiro teremos:

Negação Explícita e Implícita

Regra Geral: A negação introduzida pela partícula não (negação explícita) é traduzida pelo operador *não*, colocado imediatamente abaixo da quantificação introduzida pelo sujeito. Entretanto, se o determinador do sujeito é *cada*, *todo*, *todos os* e *cada um dos*, o operador *não* se aplica a todo o enunciado. Esta regra pode limitar a interpretação de sentenças.

As sentenças

Muitos guerreiros não conheceram Ítaca.

e

Todos os guerreiros de Ulisses não viveram.

são assim representadas:

Dahl mostra que a negação pode ser implícita, desde que nenhuma porção do domínio da quantificação satisfaça a sentença envolvida (o que da mesma forma acontece na língua portuguesa). Ela apresenta dois casos onde o determinador *nenhum* coexiste com uma negação explícita. Esses casos, denominados de falsa dupla negação, requerem um quantificador diferente ou, então que a negação seja representada duas vezes:

- na posição de sujeito (invertido com o verbo), *nenhum* é trocado por *todo*, como na sentença Não voltou nenhum guerreiro:

- em outra posição, *nenhum* é trocado pelo artigo indefinido. Na sentença Ulisses não tem nenhuma filha, temos:

Cláusulas Relativas Restritivas

Para a cláusula relativa, a regra de Colmerauer diz para tratá-la como uma sentença comum. O pronome relativo é substituído por variáveis apropriadas e a cláusula relativa é ligada à tradução do substantivo pela conjunção *e*.

Por exemplo, O cíclope que Ulisses cegou era filho de Poseidon, é representada assim:

Colmerauer salienta que a regra impede a colocação da cláusula relativa acima do *e* que a liga ao substantivo.

Embora essa regra leve a interpretações corretas, encontramos certas sentenças cuja interpretação contradiz sua aplicação. Saint-Dizier [Saint-Dizier-85] mostra um exemplo:

Uma das peças que compõem toda máquina é feita de ouro.

A interpretação correta para essa sentença é dada quando a quantificação toda, introduzida na cláusula relativa, governa a quantificação introduzida por uma das.

5.2 Tratamento de Determinadores no Sistema Chat-80

5.2.1 Determinadores e Representação do Significado das Sentenças

Pereira e Warren desenvolveram seus trabalhos seguindo [Dahl-81] e [Colmerauer-77] no problema de interpretar determinadores. A preocupação em interpretar corretamente determinadores aparece em [Pereira-82]:

É surpreendente que, para as classes básicas do Inglês como os determinadores e, mesmo dentro dos diferentes campos de estudo da linguagem representados pela lingüística, filosofia e inteligência artificial, não exista, ainda, nenhuma solução permanente para esse problema.

No sistema Chat-80, o significado de uma pergunta é dado por uma representação lógica (forma lógica). Cada determinador é traduzido em uma *quantificação*. Esta introduz uma variável lógica que liga dois predicados a ela relacionados que são chamados *domínio* e *escopo* e indicados por R e S.

Em geral, uma variável corresponde a um sintagma nominal. O domínio da variável é a tradução das palavras que compõem o sintagma nominal e, o escopo da variável é a tradução do resto da sentença.

Vejamos os determinadores usados no sistema Chat-80 e sua tradução em lógica de primeira ordem estendida:

um(a), uns(mas), algum(a), alguns(mas), o(a)	$\exists$	$exists(X, R \& S)$
--	-----------	---------------------

utiliza-se a quantificação existencial da lógica clássica. Para $exists(X,P)$ lê-se: existe algum X tal que P é verdadeiro.

nenhum(ma)	$\neg \exists$	$exists(X, R \& S)$
------------	----------------	---------------------

utiliza um tipo de negação que não é a negação padrão da lógica de primeira ordem. Para $\neg \exists P$ lê-se: não pode ser mostrado que P. Significa não provabilidade, isto é, $\neg \exists P$ é considerado verdadeiro se P não é deduzível dos fatos e regras de um domínio de aplicação particular.

$$\boxed{\boxed{\text{todo}(s)} \parallel \boxed{\setminus + \text{exists}(X, R \& \setminus + S)}}$$

aqui também é usada a não provabilidade.

$$\boxed{\boxed{\text{os, as}} \parallel \boxed{\text{exists}(X, \text{setof}(X, R, X) \& S)}}$$

para o determinador *os* foi implementada uma extensão do PROLOG chamada *setof(X,P,S)*, onde se lê: o conjunto dos X's tal que P é provável é S, onde S é não vazio.

$$\boxed{\boxed{\text{um, dois, ... N}} \parallel \boxed{\text{numberof}(X, R \& S, N)}}$$

para os numerais foram definidos predicados da forma *numberof(X,P,N)*, onde o número de X's tal que P é provável, é N. Sua definição é: $\text{numberof}(X, P, N) \Leftarrow \text{setof}(X, P, S) \& \text{sizeof}(S, N)$, a qual lê-se $P \Leftarrow Q$ como: *P se Q* (*sizeof* fornece o número de elementos no conjunto).

qual(is), que	$\text{answer}(X) \Leftarrow R \& S$
quantos(as)	$\text{answer}(N) \Leftarrow \text{numberof}(X, R \& S, N)$

para a tradução de sentenças interrogativas identificativas lê-se, $\text{answer}(X) \Leftarrow P$ como: *X é uma resposta se P*; ou, *eu desejo conhecer X se P é verdadeiro*. As variáveis no predicado *answer* são universalmente quantificadas e se lê: para qualquer X, para qualquer Y, etc.

5.2.2 Escopo das Quantificações

No sistema Chat-80 a tradução de uma sentença de entrada para uma forma lógica é feita em três passos: análise sintática, interpretação e definição do escopo de determinadores. É nesse último passo que precisamos conhecer o escopo relativo de operadores como determinadores, negação e conjunções.

As regras de escopo estão baseadas no sistema Lunar [Woods-86]. Elas são mais precisas que as propostas por Colmerauer e implementadas no sistema de Dahl. As regras de Colmerauer determinam o escopo dos operadores com base nas relações predicação-argumentos (verbos, nomes e adjetivos são representados por

predicados que possuem um ou mais argumentos), e usam apenas o conhecimento sintático e as heurísticas léxicas para a negação.

Veremos a seguir um pequeno conjunto de regras para que operadores governem outros operadores (heurísticas léxicas). Quando o escopo não puder ser decidido por essas regras, usamos a própria ordem de entrada dos operadores na sentença.

Precedência dos Determinadores

No sistema Chat-80, as regras de escopo assumem como *default* o escopo dado aos determinadores de acordo com sua ordem na árvore de quantificadores (Quant tree). Nessa árvore os quantificadores tem a sua ordem de entrada preservada; mas, existe um conjunto pequeno de exceções a essa regra *default*.

A razão dessas regras de exceção é o fato dos operadores usados não serem, em geral, suficientes para derivar uma única escolha de escopo de quantificadores. Por exemplo, a sentença abaixo [Pereira-83]:

Um embaixador visitou todos os países europeus.

pode ser lida distributivamente, com possivelmente diferentes embaixadores visitando países diferentes — quando o determinador *um* está dentro do escopo de *todos os* —; ou, coletivamente, com um único embaixador visitando todos os países europeus quando o determinador *todos os* está dentro do escopo de *um*.

Todas as regras de exceção podem ser parafraseadas por: *se um determinador A aparece acima de um determinador B na Quant tree, então dê a B maior escopo que A (o que será contrário ao default). B então governa A.*

Essas regras também são usadas para especificar o comportamento de determinadores em relação a outros operadores como a negação e os pronomes interrogativos.

As regras não conseguem fazer distinções refinadas por não utilizarem nenhuma informação pragmática acerca das palavras usadas.

No sistema Chat-80 os determinadores são divididos em duas classes: *fortes* e *fracos*. Qualquer determinador forte governa outro fraco. São determinadores fortes *cada* e *qualquer/algum/nenhum* (traduções de *any*); todos os outros são fracos.

Um determinador forte irá mover-se acima e à esquerda na *Quant tree* até encontrar outro determinador forte. Quando uma *Quant tree* não tem mais movimentos para cima ou à esquerda, ela está na sua posição final.

Analisemos as regras de exceção. Nas seguintes frases:

a população de cada país
a população de (any) qualquer país

as regras dão ao determinador de país um escopo maior do que ao determinador de população, tornando o valor de população dependente do de país. Mas, as regras *default* dão ao determinador de país, em:

a população de todo país

um escopo mais estreito que o determinador de população. Entretanto, o outro escopo pode ser utilizado usando *cada* ao invés de *todo*. É assim que Pereira consegue um modo de expressar a quantificação universal com escopo estreito, o que de outra forma não seria possível.

Negação e o Determinador *Any* (Qualquer/Nenhum/Algum)

A única regra de exceção com respeito à negação é que *any* governa a negação.

O determinador *any* pode ser visto como um quantificador existencial de escopo estreito, ou, como um quantificador universal com largo escopo relativo a negação.

Ambas as interpretações fornecem leituras equivalentes em sentenças declarativas ou interrogativas identificativas com um verbo principal explicitamente negado. Todavia, elas apresentam problemas em sentenças sem negação explícita. Na sentença:

Pedro gosta de (any) qualquer garota.

a tradução para o quantificador existencial com escopo estreito produz:

$$(\exists G)(\text{garota}(G) \& \text{gosta}(\text{Pedro}, G))$$

que não é aceitável. A tradução para o quantificador universal de escopo largo produz uma interpretação aceitável:

$$(\forall G)(\text{garota}(G) \Rightarrow \text{gosta}(\text{Pedro}, G))$$

Para sentenças interrogativas verificativas, o uso de *any* como quantificador universal de escopo largo gera uma boa tradução se considerarmos o operador sim-não (mostra que a sentença é interrogativa verificativa) com as mesmas propriedades que a negação.

Em Pedro gosta de (any) qualquer/alguma garota? deveremos traduzir para:

$$(\forall G)(\text{garota}(G) \Rightarrow (\text{gosta}(\text{Pedro}, G) \Rightarrow \text{answer}(\text{sim})))$$

que é logicamente equivalente à leitura intuitiva:

$$\text{answer}(\text{sim}) \Leftarrow (\exists G)(\text{garota}(G) \& \text{gosta}(\text{Pedro}, G))$$

Pereira comenta que, para sentenças interrogativas identificativas sem um verbo principal negado e, para as sentenças interrogativas verificativas com um verbo negado, o uso de *any* (*qualquer* para a primeira e *nenhum* para a segunda) pode ser considerado agramatical. Como por exemplo:

Quais países fazem fronteira com (any) qualquer oceano?

Sentenças Interrogativas Identificativas e o Determinador *Cada*

A pragmática de sentenças interrogativas identificativas requer uma resposta que inclua os valores de cada sintagma nominal. Na sentença abaixo:

Quais países fazem fronteira com o Brasil?

temos a tradução para:

$$\text{answer}(C) \Leftarrow \text{pais}(C) \& \text{faz_fronteira}(C, \text{brasil})$$

O determinador *cada* assume um papel distributivo, governando muitos determinadores inclusive os pronomes interrogativos; e, o quantificador universal que o traduz é omitido (como é usual na lógica clausal). Portanto, uma resposta a uma pergunta contendo *cada* deverá ser uma lista indexada de valores, um para cada entidade satisfazendo as condições do sintagma nominal com tal determinador.

Traduzimos:

Quantos países existem em cada continente?

para:

$$\text{answer}(X-N) \Leftarrow \text{continente}(X) \&\text{numberof}(C, \text{pais}(C) \&\text{em}(C, X), N)$$

e temos como resposta:

africano-48, americano-31, asiático-39, australasiano-6, europeu-32

Cláusulas Relativas

Como Colmerauer, também Pereira e Warren assumem que a nenhum determinador de um modificador (por exemplo em uma cláusula relativa) é permitido ter escopo maior que a sentença modificada. Embora seja verdade para muitas sentenças, existem algumas que contradizem essa regra (como foi visto no item 5.1.2).

5.3 Determinação do Escopo de Quantificadores segundo McCord

Discutimos a seguir como McCord [McCord-82] determina o escopo semântico de modificadores, em especial o escopo de quantificadores.

O sistema de McCord analisa a sentença de entrada em três estágios. O primeiro produz a análise sintática em um formato especial — item sintático ou *syn* — e o segundo e o terceiro traduzem este formato para uma forma lógica que, então, faz a interação com a base de dados.

5.3.1 Modificadores, Estrutura Sintática e Interpretação Semântica

Um item sintático é uma estrutura de dados uniforme para todos os tipos de sentenças. A forma de um item sintático é:

syn(*Características*, *Determinador*, *Predicação*, *Modificadores*).

onde:

- *Características* é uma lista de átomos contendo informações sobre a forma morfológica do item.
- *Determinador* é um termo que contém informações sobre o modo pelo qual um dado item sintático pode ser usado como um modificador de outros itens.
- *Predicação* é a predicação central de um item, representando, por exemplo, o significado de um verbo em um sintagma verbal ou o núcleo de um sintagma nominal.
- *Modificadores* é a lista de todos os modificadores de um dado item ordenados pela entrada. Cada modificador é novamente um *syn*.

A noção tradicional de determinador foi generalizada por McCord. Todo item sintático tem um determinador; não somente sintagmas nominais formados de substantivos comuns, mas também os formados por substantivos próprios, por pronomes, por preposições, cláusulas relativas, etc. Por exemplo, a estrutura sintática da sentença:

Cada homem viu Pedro.

é a seguinte:

[s,dcl] principal ver(X,Y,C)

[sn,sg] todo:X homem(X)

[adv] conjuncao passado(C)

[sn,sg] def(sg):Y Y = Pedro

sendo que *s* significa sentença; *dcl* declarativa; *sn* sintagma nominal; *sg* singular; *adv* modificador adverbial. Cada linha dá as características, o determinador e o núcleo de um *syn*. Os modificadores de um *syn*, se existir algum, são listados abaixo e à direita do *syn*.

A interpretação semântica da sentença acima — na forma lógica — é a seguinte:

$$dcl(homen(X) \Rightarrow Y = Pedro \& ver(X, Y, C) \& passado(C))$$

O sistema possui dezesseis determinadores. Todos eles, à exceção de *principal*, que é tratado de uma forma especial, são discriminados abaixo:

1. *todo*:*X* — Usado para sentenças quantificadas universalmente.
2. *def*(*Num*):*X* — Para sentenças definidas; *Num* pode ser *sg* ou *pl*, indicando singular ou plural.
3. *indef*:*X* — Usado para a maioria das sentenças existenciais.
4. *indef1*:*X* — Para certas sentenças existenciais de largo escopo.
5. *int_identificativa*(*Tipo*):*X* — Para sentenças interrogativas identificativas. Produz um quantificador-pergunta que indica uma requisição para encontrar *X*. *Tipo* pode ser *principal* (para *quantos*) ou *def* ou *indef*.
6. *num*(*N*):*X* — Lê-se *exatamente N X*,
7. *mais**que*(*N*):*X* — Lê-se *mais do que N X*,
8. *menos**que*(*N*):*X* — Lê-se *menos do que N X*,
9. *nomínimo*(*N*):*X* — Lê-se *no mínimo N X*,
10. *nomáximo*(*N*):*X* — Lê-se *no máximo N X*,
11. *nenhum*:*X* — Lê-se *não existe X*.
12. *conjunção* — Usado para certos modificadores adjetivais e adverbiais, para cláusulas relativas, etc.
13. *subst*:*X* — Faz a predicação de um modificador ser substituída (unificada) pela variável *X*.

14. *aplicarpara*(Y) — Oposto de *subst:X*. O modificando é unificado com a variável Y que aparece na predicação do modificador.
15. *id:X* — Identidade. O resultado da modificação é o modificando inalterado.

McCord argumenta que a razão de se ter um nível sintático separado é o escopo de modificadores:

Se a hierarquia semântica de modificadores seguisse exatamente sua hierarquia sintática, então seria mais conveniente especificar a interpretação semântica diretamente nas regras da gramática. Todavia, é sabido que a linguagem natural não segue este esquema simples.

Em seu trabalho, modificadores são complementos ou adjuntos.

5.3.2 Hierarquia Sintática *versus* Hierarquia Semântica

O escopo de um modificador pode depender de dois fatores:

- o *conteúdo* de um particular modificador; e,
- a *posição* do modificador.

Para as sentenças:

Cada professor ensina um curso.
Cada professor ensina um certo curso.

Temos a representação mais provável para a primeira sentença quando a quantificação introduzida pelo modificador *cada* tem escopo maior, isto é, domina a quantificação do modificador *um*. Para a segunda sentença temos a representação mais provável quando a quantificação do modificador *um certo* tem escopo maior que a do modificador *cada*.

No sistema de McCord isto é solucionado atribuindo a um curso o determinador *indef:X* e a um certo curso o determinador *indef1:X* que têm precedência maior que *todo:X*.

Os sintagmas nominais que começam com *algum particular* e *qualquer (any)* também possuem o determinador indefl:X.

A hierarquia semântica dos modificadores pode violar relações verticais e horizontais na árvore sintática.

Um filho de um dado modificador M pode ter escopo maior que o próprio M. Na sentença:

Dê-me a média de cada estudante.

o complemento nominal *cada estudante* é filho de *a média de cada estudante* e tem o determinante todo:_, o qual possui escopo maior que o sintagma nominal *a média* com o determinador def(sg):_.

McCord faz críticas a Colmerauer e Dahl por adotarem a regra que dá à quantificação do complemento de um nome, um escopo maior que a quantificação gerada pelo nome (o que nem sempre produz interpretações corretas). McCord exemplifica sua crítica com as sentenças abaixo:

Toda requisição para um aumento de nota será recusada.
A procura por uma resposta foi impedida pela sonolência.

5.3.3 Determinação da Hierarquia Semântica de Modificadores

O sistema de McCord realiza a interpretação semântica em dois estágios: o primeiro rearranja a árvore sintática de modo que a nova estrutura reflita a hierarquia semântica desejada para os modificadores. O segundo estágio desempenha as ações implícitas na árvore rearranjada.

A operação que rearranja a árvore sintática usa dois movimentos de nós: o horizontal e o vertical. Os movimentos horizontais dos nós estão baseados na atribuição de precedência para cada item sintático. A relação entre um item sintático e sua precedência é a seguinte:

- $syn(-, int_identificativa(-) : -, -, -) \rightarrow 10$
- $syn([int_verificativa], -, -, -) \rightarrow 10$

- $\text{syn}(-, \text{indef1} : X -, -, -) \rightarrow 5$
- $\text{syn}(-, \text{aplicarpara}(-), -, -) \rightarrow 5$
- $\text{syn}(-, \text{nenhum} : -, -, -) \rightarrow 5$
- $\text{syn}(-, \text{def}(-) : -, -, -) \rightarrow 3$
- $\text{syn}(-, \text{todo} : -, -, -) \rightarrow 3$
- $\text{syn}([\text{adv}], -, -, -) \rightarrow -1$ (modificadores adverbiais)
- $\text{outros_tipos_de_itens} \rightarrow 0$

Se um nó N1 tem um número de precedência maior que um nó irmão N2, ele aparecerá à esquerda do nó N2 na reordenação. Quando os nós têm números de precedência iguais, a ordem original da esquerda para direita é preservada.

Os movimentos verticais dos nós (que se promovem de filhos para uma relação de irmãos) são realizados pela relação *acima(Nó, Filho)*. Esta operação de promoção é uma ação *default* que tende a linearizar a árvore, pois, os filhos são promovidos possivelmente através de vários níveis, até que uma relação *acima* os impeça de subir.

5.4 Considerações de Allen sobre o Escopo de Quantificadores

Em seu livro, *Natural Language Understanding* [Allen-87], Allen discute as características sintáticas e semânticas que afetam o escopo de quantificadores e operadores opacos em sentenças da linguagem natural.

Allen observa que nem todos os escopos podem ser determinados independente do contexto. Mas, por outro lado, diz que certas sentenças tem uma forte preferência a um particular escopo.

Muitas das heurísticas utilizadas por Allen para a decisão dos escopos de quantificadores são as definidas em [Woods-86] e [Pereira-83]. Ele tece considerações sobre o escopo de quantificadores em sentenças com sintagmas nominais simples, sintagmas preposicionais, cláusulas relativas e a interação com negações. Também discute outros operadores que envolvem ambigüidade de escopo, tais como operadores opacos, que não serão vistos aqui.

5.4.1 Sentenças com Sintagmas Nominais Simples

Usando a posição sintática do determinador na sentença e o poder de dominação do determinador, Allen chega a seguinte lista inicial de preferências por escopos maiores:

$o(s), a(s) > cada > pronomes_interrogativos > todo, todos\ os, algum, um$

Os determinadores *cada* e *todo* ou *todos os* parecem ser mapeados para o quantificador universal, mas, diferem nas dependências que eles criam. Por exemplo:

Um homem viu todos os cachorros.

Um homem viu cada cachorro.

A interpretação para a primeira sentença será: existe um único homem que viu todos os cachorros. A interpretação para a segunda sentença não está clara, pois, pode existir diferentes homens para cada cachorro.

Assim, *cada* tende a indicar um escopo maior que o determinador *todo* ou *todos os*.

Levando-se em consideração a posição sintática do determinador na sentença, parece existir uma forte tendência para o primeiro sintagma nominal dominar os outros. Vejamos as sentenças abaixo:

Um homem viu todos os cachorros.

Todos os cachorros viram um homem.

Na primeira sentença, um tem escopo maior que todos os, o que não ocorre na segunda. Esta tendência leva a uma diferenciação entre as vozes ativa e passiva, como Colmerauer [Colmerauer-77] também salienta. O exemplo:

Um homem foi visto por todos os cachorros.

fornece a idéia de que existe um único homem.

O artigo definido possui um escopo maior, independente de sua posição em sentenças com sintagmas nominais simples, como em:

Todos os cachorros viram o homem.
O homem viu todos os cachorros.
Cada cachorro viu o homem.

Assim, sintagmas nominais com o determinador *o(s)* tem escopo maior, seguidos por sintagmas nominais com *cada*, e finalmente, por sintagmas nominais com *todo* e *um*.

Sintagmas nominais com pronomes interrogativos parecem estar entre *cada* e *todos os*. Em:

Quem viu todos os cachorros?
Quem viu cada cachorro?

temos que: na primeira sentença estamos pedindo uma única pessoa, mas na segunda sentença podemos pedir por uma lista de pessoas, isto é, uma ou mais para cada cachorro.

Um algoritmo inicial para sentenças com sintagmas nominais simples seria:

1. Dê a todos os sintagmas nominais com artigo definido o maior escopo.
2. Em sentenças com a forma *Existe Q X que ...* dê ao termo *X* o próximo maior escopo. (*Q* pode ser *um*, *algum*, *um pouco*, etc.)
3. Dados dois determinadores *Q1* e *Q2* com escopo relativo ainda não determinado pelos passos antecedentes, se *Q1* > *Q2* na lista de preferências por maiores escopos, torne *Q2* dependente de *Q1*, a menos que *Q1* seja o primeiro constituinte da sentença.

Este algoritmo não consegue decidir no caso de existir conflito entre a posição sintática e o poder de dominação do determinador em questão. Para exemplificar temos:

Quem viu cada cachorro?

5.4.2 Sentenças com Sintagmas Preposicionais

Allen mostra que existem sentenças onde, mesmo um sintagma nominal composto por um artigo definido pode se tornar dependente de um sintagma preposicional. Por exemplo:

A cozinha em toda casa tem um fogão.

A interpretação mais adequada é aquela em que existem muitas cozinhas diferentes; isto é, com o determinador *toda* tendo escopo maior que o determinador *a*.

Assim, para manusear sintagmas preposicionais, um novo passo deverá ser adicionado ao algoritmo:

Se um sintagma nominal X é modificado por um sintagma preposicional Y então X é dependente de Y.

5.4.3 Sentenças com Cláusulas Relativas

Segundo Allen, cláusulas relativas parecem afetar o escopo exatamente de modo oposto a sentenças com sintagmas preposicionais. Por exemplo, na sentença:

A cama que eu fiz todo dia quebra.

a interpretação mais viável é aquela em que existe uma única cama.

Uma forma de examinar a causa dessa preferência de escopo é considerar a interpretação semântica de cláusulas relativas como um processo separado da sentença na qual está aninhada. Os determinadores na cláusula relativa não têm, geralmente, maior escopo que a própria cláusula.

O algoritmo final que inclui essas considerações é:

1. Considere recursivamente todas as cláusulas relativas e, então, torne cada *Q* introduzido na cláusula relativa dependente do sintagma nominal que a cláusula relativa modifica.

2. Dados dois determinadores Q_{sn} e Q_{sp} , onde Q_{sp} é um modificador do sintagma nominal de Q_{sn} , torne Q_{sn} dependente de Q_{sp} .
3. A menos que seja contradito pelos passos 1. ou 2., dê a todos os sintagmas nominais com artigos definidos o maior escopo.
4. A menos que seja contradito pelos passos antecedentes, em sentenças com a forma *Existe Q X que ...* dê ao termo X o próximo maior escopo. (Q pode ser *um, algum, um pouco*, etc.)
5. Dados dois determinadores $Q1$ e $Q2$, cujo escopo relativo ainda foi não determinado pelos passos antecedentes, se $Q1 > Q2$ na lista de preferências por maiores escopos torne $Q2$ dependente de $Q1$, a menos que $Q1$ seja o primeiro constituinte da sentença.

5.4.4 Negação

O modo como a negação interage com determinadores pode ser analisado como um problema de escopo, pois uma mudança do escopo da negação muda também as interpretações de uma sentença. A sentença:

Todo pato não é preto.

para muitas pessoas, é ambígua entre o caso onde nenhum pato é preto:

$$\forall p(\text{pato}(p) \Rightarrow \text{nao preto}(p))$$

e o caso onde alguns patos não são pretos, mas alguns sim:

$$\text{nao}(\forall p(\text{pato}(p) \Rightarrow \text{preto}(p)))$$

Se a negação não modifica a sentença, o número de possíveis interpretações pode ser pequeno. Por exemplo, a sentença:

Nem todo garoto gosta de Maria.

permite somente a interpretação *alguns garotos não gostam de Maria, mas alguns sim*.

Isto leva a negação a ter um escopo maior que outros determinadores. Ainda assim, temos um problema quando há vários determinadores. Como na frase:

Nem todo garoto ama alguma garota.

Como antes, a negação deverá ter maior escopo que o quantificador universal; mas, temos ainda várias interpretações dependendo do escopo do quantificador existencial. Assim, para a sentença acima, temos as interpretações:

- existe um garoto que não ama nenhuma garota (aninhamento: não, todo, alguma); e,
- existe uma certa garota que nem todo garoto ama (aninhamento: alguma, não, todo).

O que não podemos ter é a interpretação onde não é o caso de uma particular garota ser amada por todo garoto (aninhamento: não, alguma, todo). Pois nenhum outro quantificador pode vir entre a negação e o universal.

Capítulo 6

Escopo de Quantificadores: Geração de Vários Escopos

Neste capítulo veremos trabalhos onde o tratamento dado aos determinadores é mais flexível. Começamos com o de Saint-Dizier [Saint-Dizier-85], já citado no capítulo 5, que apresenta um sistema de reescrita para gerar as várias leituras relevantes de uma sentença em linguagem natural. O trabalho é tratado em detalhe, pois algumas idéias nele contidas serão usadas no módulo pragmático para tirar ambigüidade de escopo de quantificadores.

Os outros trabalhos são o de Pereira [Pereira-86], o de Hobbs e Shieber [Hobbs-87], e o Sistema TEAM [Grosz-87].

O de Pereira gera todas as permutações possíveis de escopos dos quantificadores, com representações sem variáveis livres — fórmulas bem formadas fechadas. O trabalho de Hobbs e Shieber gera todos os escopos possíveis, eliminando somente aqueles que são lingüísticamente inaceitáveis em qualquer contexto. E, finalmente, o Sistema TEAM que apresenta uma interface em linguagem natural transportável.

Ainda sobre o sistema TEAM, veremos como ele determina o escopo de quantificadores, as suas críticas ao conjunto de formas lógicas alternativas, a sua interação com o usuário e o uso de informações pragmáticas para derivar a leitura correta da sentença.

6.1 O Sistema de Reescrita de Saint-Dizier

Saint-Dizier [Saint-Dizier-85] parte da representação semântica baseada na lógica — árvore quantificada de três ramos — formulada por [Colmerauer-77] e utilizada por [Dahl-81]. Ele comenta que seu trabalho pode ser aplicado a outros formalismos, como por exemplo o de [Pereira-83].

O objetivo de seu trabalho é determinar um conjunto de regras que possam produzir:

- a representação lógica correta de uma sentença, do ponto de vista da ordem dos quantificadores; e,
- todas as interpretações relevantes de uma sentença ambígua.

Por uma questão didática, iremos esclarecer os conceitos de sistema de reescrita, domínio (*range*), indicador de domínio (*range indicator*) e escopo (*scope*) e, analisar e criticar o sistema de reescrita de Saint-Dizier.

Sistema de Reescrita

Seja A um alfabeto e P um conjunto finito de pares ordenados de palavras em A . Um sistema de reescrita é um par ordenado (A, P) . Os pares (a, b) de P são chamados regras de reescrita ou regras de produção e são denotados por $a \rightarrow b$.

Domínio e Escopo

Embora Saint-Dizier mencione *range of quantifiers* e *scope of quantifiers* ele não deixa claro se vai usá-los no mesmo sentido de Pereira [Pereira-82], [Pereira-83]. O uso impreciso dessas expressões no artigo de Saint-Dizier, levou-nos a caracterizar *range* (domínio) e *scope* (escopo) da forma mais adequada para nosso uso.

O domínio de quantificadores é a expressão lingüística que corresponde ao conjunto onde a variável — letra-argumento do quantificador — é interpretada.

Dizemos que um domínio de um quantificador X é maior que outro Y se a expressão X precede Y na representação semântica formal dos significados das sentenças. Como na sentença:

Um rio corre através de cada país europeu.

Que pode ser lida distributivamente como diferentes rios correndo através de diferentes países europeus; ou coletivamente, como um único rio correndo através de todos os países europeus.

No primeiro caso, o quantificador introduzido por *um* está dentro do escopo do quantificador introduzido por *cada*, e a' i dizemos que o *domínio* de *cada* é maior que o *domínio* de *um*.

A expressão escopo é usada do mesmo modo que em lógica de primeira ordem.

6.1.1 Definição e Construção do Indicador de Domínio

O indicador de domínio, segundo o próprio, é uma noção artificial sem embasamento lingüístico. Para sua construção, todos os determinadores e negações que aparecem na sentença de entrada são indexados de acordo com sua ordem de aparecimento — deve-se notar que a ordem de entrada dos determinadores não se altera. O indicador de domínio resulta da substituição do determinador ou da negação pelo quantificador que ele introduz.

Por exemplo, se em uma sentença de entrada aparecessem os determinadores: O ... um ... todos os, eles seriam indexados assim:

O1 um2 todos3 os3

O autor, inspirado nos quantificadores da lógica de primeira ordem, particiona os determinadores em quatro classes, fazendo-as denominar os quantificadores. Estas classes são as seguintes: quantificador distributivo universal (D), quantificador existencial da lógica de primeira ordem (E), quantificador existencial quantitativo (Q) e quantificador universal da lógica de primeira ordem (U):

$D = \{\text{todo, cada}\}$

$E = \{\text{um(a), algum(a), alguém, muitos, ninguém, muitos dos, poucos, que, quem, qual, a(o) qual, o, ...}\}$

$Q = \{\text{uma porção, um pouco, ...}\}$

$U = \{\text{os, todos os, todos, nenhum, ...}\}$

Esta partição está intimamente ligada às regras de reescrita para a reorganização do indicador de domínio.

Saint-Dizier não fornece nenhuma razão especial para esta partição, diz somente que os determinadores são classificados de forma a construir interfaces com uma linguagem natural simples e direta, detectando e resolvendo as ambigüidades de escopo de quantificadores.

O indicador de domínio é o resultado da substituição de um determinador pelo quantificador que ele introduz. Assim, para uma sentença composta pelos determinadores *um* e *todo* que, depois de indexados, tornam-se:

um1 todo2

o indicador de domínio será: E1 D2.

6.1.2 Aplicação das Regras de Reescrita sobre o Indicador de Domínio

O objetivo das regras de reescrita é gerar todas as configurações relevantes de quantificadores a partir do indicador de domínio inicial. A forma geral de uma regra R_i é:

$$R_i : X_n Y_m \rightarrow Y_m X_n (\text{ou } X_m Y_n)$$

onde X, Y pertencem a $\{E, D, Q, U\}$ e n, m são inteiros tal que $n < m$.

Seja n, m, p inteiros tal que $n < m < p$, tem-se as seguintes regras:

$$R_1 : U_n E_m \rightarrow E_m U_n$$

$$R'_1 : U_n E_m \rightarrow U_m E_n$$

$$R_2 : E_n D_m \rightarrow D_m E_n$$

$$R_3 : Q_n U_m \rightarrow Q_m U_n$$

$$R_4 : E_n U_m \rightarrow E_m U_n$$

$$R_5 : D_n E_m \rightarrow E_m D_n$$

$$R'_5 : D_n E_m \rightarrow D_m E_n$$

$$R_6 : Q_n D_m \rightarrow D_m Q_n$$

$$R_7 : NAO_n X_m \rightarrow X_m NAO_n$$

onde $X = D, E$ ou U .

$$R_8 : U_n NAO_m \rightarrow NAO_m \text{algum}_n$$

$$R_9 : U_n NAO_m E_p \rightarrow E_p U_m NAO_n$$

Essas regras são aplicadas da esquerda para a direita no indicador de domínio até não existirem mais regras a serem aplicadas. A ordem em que as regras são aplicadas não interfere no processo de reordenação do indicador de domínio. Para sentenças não ambíguas, essas regras produzem uma única configuração de quantificadores.

A crítica que fazemos ao trabalho de Saint-Dizier refere-se ao seu sistema de reescrita, pois, este conjunto de regras nem sempre é adequado. Além do mais, o autor não justifica a falta de algumas regras, como a que daria a interpretação correta para a sentença:

Quais notas obtiveram todos os estudantes?

O sistema deveria fornecer a interpretação de que cada aluno possui uma dada nota. Entretanto, através da regra 4, a interpretação é a de que há uma única nota para todos os alunos.

6.1.3 Exemplos de Aplicações das Regras

Seja a sentença:

Um operário coleta todas as fitas magnéticas.

O seu indicador de domínio é: $E_1 U_2$; a regra R_4 é aplicada e o indicador de domínio é reescrito como $E_2 U_1$. Nenhuma outra regra é aplicável e sua representação final é:

$\text{um}(\text{x}, \text{operário}(\text{x}), \text{todas_as}(\text{y}, \text{fita_magnética}(\text{y}), \text{coleta}(\text{x}, \text{y})))$,

ou:

Conjunções

Para as conjunções são introduzidos parênteses no indicador de domínio. Os parênteses são indexados como se fossem um quantificador comum. Como em:

Todos os professores e alguns estudantes tem um microcomputador.

Onde o indicador de domínio é $(UE)_1E_2$.

Quando as regras de reescrita são aplicadas, os quantificadores parentizados são considerados como um único determinador através das regras:

$(X_1 \dots X_n)_j$ é equivalente a Y_j e seu valor é:

X se todos os X_k são o mesmo identificador,

D se existe um $X_k = D$ (a distributividade de D é estendida aos outros determinadores)

U se existe um $X_k = U$ e nenhum $X_j = D$

E caso contrário.

O indicador de domínio do exemplo acima é equivalente a U_1E_2 . As regras R_1 e R'_1 são aplicáveis.

A aplicação da regra R'_1 dá a representação correta (U_2E_1) pois um micro-computador é frequentemente um computador pessoal, o que não ocorre em:

Todos os professores e alguns estudantes usam um VAX.

O indicador de domínio é ainda U_1E_2 , mas neste caso a aplicação da regra R_1 dá a representação correta, pois um VAX é um computador compartilhado. Esta última sentença é assim representada:

Para conjunções de sintagmas nominais, onde alguns deles apresentam complementos nominais, o autor fornece duas regras:

1. Somente o primeiro determinador de cada sintagma nominal (na ordem de entrada da sentença) é levado em conta para encontrar o quantificador equivalente de todos os determinadores de sintagmas nominais ligados por conjunção.
2. O sistema de reescrita é aplicado separadamente sobre cada sintagma nominal e, depois, sobre o indicador de domínio total encontrado pela regra 1.

Para um melhor entendimento dessas regras, tomemos a sentença:

Alguns professores em toda escola e muitos dos homens de negócio têm um microcomputador.

Seu indicador de domínio é $((E_1 D_2) E_1)_1 E_2$. Aplicando-se a regra 1 temos $(EE)_1 E_2$ e, então, $E_1 E_2$ como indicador de domínio total.

Pela regra 2 o sistema de reescrita deve ser aplicado primeiro em $E_1 D_2$ e depois no indicador de domínio total para conseguir o indicador de domínio final:

$$((D_2 E_1) E_1)_1 E_2$$

A representação semântica formal para a sentença acima é a seguinte:

Cláusulas Relativas

O autor somente considera as cláusulas relativas que introduzem restrições ao conjunto de objetos denotados pelo nome ao qual se referem. Do mesmo modo que [Dahl-81], na representação semântica, a cláusula relativa é ligada ao nome

que ela se refere pela conjunção *e*, sendo também a hierarquia dos quantificadores na cláusula relativa processada independentemente da cláusula principal.

Como na sentença:

Eu conversei com um homem de
cada país sul-americano.

Onde a aplicação da regra R_2 dá escopo maior a cada. Entretanto, na sentença:

Eu conversei com um homem que estava representando
cada país sul-americano.

O determinador *cada* está dentro do escopo de *um*, e a sentença fica representada assim:

Sentenças Interrogativas

Sentenças como:

Quais rios correm através de cada país europeu?

requerem múltiplas respostas. Em particular, a frase acima pode ser parafraseada assim:

Para cada país, encontre todos os rios que correm por ele

O sistema de reescrita de Saint-Dizier leva em conta este fato e, como em [Pereira-82], uma resposta à uma pergunta contendo *cada* deverá ser uma lista indexada de valores, um para cada entidade que satisfaça as condições do sintagma nominal com aquele determinador.

O indicador de domínio de Quais rios correm através de cada país europeu? é E_1D_2 . Aplicando-se a regra R_2 temos a representação:

$$\text{PERG}((x,y), \\ \text{cada}(x,e(\text{país}(x),\text{europeu}(x)),\text{quais}(y,\text{rio}(y),\text{corre_através}(y,x))))$$

Nota-se aqui que um determinador pode estar acima de um pronome interrogativo. O operador PERG indica que temos uma sentença interrogativa.

6.1.4 Informação Contextual *versus* Paráfrases

Se uma sentença é ambígua, sua representação formal *correta* pode ser determinada de duas formas:

- com ajuda do usuário através de paráfrases; e,
- através de inferências de informações contextuais armazenadas na base de conhecimento.

O uso de paráfrases — o sistema fornecendo as várias leituras possíveis a partir de uma sentença — pode desencadear alguns problemas:

- sobrecarregar o usuário com muitas sentenças; e,
- adicionar ambigüidade, o que exige clareza na formulação das paráfrases.

O uso de informação contextual seria uma alternativa em relação ao uso de paráfrases, quando o contexto é suficientemente restrito para determinar a intenção do usuário; caso contrário, o sistema deve sair da ordem natural de pergunta-resposta na tentativa de tornar clara a pergunta do usuário.

Muitos diálogos não seguem a simples ordem do par pergunta-resposta. As variações surgem quando existem mal-entendidos ou ambigüidade em sentenças anteriores. Um exemplo de diálogo para resolver a ambigüidade introduzida por uma pergunta é mostrado abaixo [Allen-80]:

1. Quando é o trem Windsor?
2. Para Windsor?
3. Sim.
4. 3:15.

A pergunta 2 é razoável porque para a sentença 1 duas situações são possíveis:

1. a pessoa que a pronunciou deseja encontrar um trem chegando *de* Windsor.
2. a pessoa deseja embarcar em um trem partindo *para* Windsor.

Veremos agora como as informações contextuais podem interagir com as regras de escopo.

Se a interface em linguagem natural serve para acessar uma base de conhecimento, as informações contextuais são um conjunto de:

- fatos armazenados na base; e,
- as relações sobre esses fatos; e,
- limites de integridade.

com a possibilidade de verificar quando a aplicação de uma regra do sistema de reescrita não contradiz qualquer fato, relação ou limite de integridade.

Seja a sentença:

Todo primeiro-ministro de um país europeu
visitou cada capital sul-americana.

A ambigüidade de escopo de quantificadores é local ao sintagma nominal *Todo primeiro-ministro de um país...* e de acordo com as regras R_5 e R'_5 , duas representações são possíveis. Com escopo largo, um país é existencial e com escopo estreito, ele é universal.

Se existe a regra:

Há somente um primeiro-ministro em cada país.

O escopo estreito para um país é a interpretação correta e somente a regra correspondente pode ser aplicada.

O problema que surge nessa busca de informação contextual é que a interação com a base pode ser difícil de manusear. Para a sentença:

Quais rios correm através de cada país europeu?

uma representação possível seria:

que deverá ser rejeitada se na base de conhecimento estiverem presentes os fatos:

1. Um país que é uma ilha não tem qualquer fronteira comum com qualquer outro país.

2. Islândia é uma ilha.

3. Chipre é uma ilha.

É possível deduzir de 1), 2) e 3) que alguns países europeus não podem ter um rio em comum com outros países europeus e, assim, a representação acima será rejeitada.

A interpretação correta para a sentença acima é a seguinte:

$$\text{perg}((x,y), \\ \text{cada}(x,e(\text{país}(x),\text{europeu}(x)),\text{quais}(y,\text{rio}(y),\text{corre_através}(y,x))))$$

Mesmo que informações contextuais não estejam disponíveis para todas as regras, elas ao menos irão reduzir o número de representações possíveis de uma sentença ambígua. Se para uma sentença, duas regras podem ser aplicadas e não existe nenhuma informação contextual disponível, o usuário deverá ser requisitado para dizer qual a formulação correta para a parte ambígua da sentença.

6.2 O Algoritmo de Pereira para a Geração de todas as Permutações de Quantificadores

O algoritmo de Pereira [Pereira-86] enfoca apenas uma pequena questão do problema da ambigüidade de escopo de quantificadores: como gerar formas lógicas para uma sentença com todas as permutações possíveis de quantificadores. Pereira comenta que este gerador pode ser usado para propor escopos alternativos a um módulo do sistema de PLN que faça críticas ao escopo. Estas críticas podem ser baseadas na estrutura sintática, ordem de entrada dos determinadores da sentença, em determinadores específicos ou qualquer outra informação relevante.

Um gerador de escopos tem que satisfazer duas propriedades principais que, segundo Pereira, são as seguintes:

1. Coerência: toda forma gerada deve ser uma forma bem formada fechada (isto é, sem variáveis livres).
2. Completude: toda permutação possível de escopo deve ser gerada.

No algoritmo de Pereira, o significado de um determinador é uma função do significado do sintagma nominal e do significado do sintagma verbal, respectivamente o domínio e o escopo do determinador. O método de Pereira para geração de escopos constroi uma representação intermediária, a árvore quantificada, mostrada abaixo:

$$q(X,D,E)$$

onde X é a expressão que representa o significado do determinador e D e E são as subárvores das quais o domínio e o escopo de X são obtidos.

Portanto, os escopos alternativos na forma lógica correspondem a escolhas alternativas do domínio e do escopo para os quantificadores da forma lógica.

O trabalho do gerador de escopos pode ser resumido assim: *considere os determinadores na sentença e gere todas as escolhas possíveis do domínio e escopo para os quantificadores correspondentes.*

Os programas PROLOG que realizam a análise sintática e a interpretação semântica, juntamente com o gerador de escopo de Pereira encontram-se no capítulo 7.

A gramática de cláusulas definidas, original, de Pereira [Pereira-86] foi adaptada para o português e estendida a fim de permitir outros tipos de sentenças.

6.3 Eliminação de Leituras Inaceitáveis no Trabalho de Hobbs e Shieber

Temos dois tipos de algoritmos geradores. Os algoritmos ingênuos, que são aqueles que geram $n!$ leituras diferentes para uma sentença com n determinadores e, os algoritmos que suprimem leituras falsas devido a estrutura de suas fórmulas lógicas. Um exemplo do tipo de algoritmo ingênuo é o gerador de escopos alternativos de [Pereira-86], visto no item 6.2, que gera às vezes mais de $n!$ leituras da sentença, pois fornece também as leituras logicamente equivalentes.

O trabalho de Hobbs e Shieber é dedicado ao estudo de um algoritmo de geração de escopos de quantificadores que não seja esbanjador, como aquele baseado na permutação de quantificadores.

O algoritmo de Hobbs e Shieber suprime as leituras que são falsas por razões puramente estruturais, isto é, por razões que seguem das relações entre as estruturas das sentenças e as estruturas das suas formas lógicas, sendo independente do significado de sentenças particulares. Portanto, esse algoritmo não utiliza conhecimento do mundo para, por exemplo, impedir que o quantificador de a altura domine o quantificador de todos os picos na sentença:

A altura de todos os picos europeus ultrapassa mil metros.

Ele também não está preocupado com redundâncias lógicas, como por exemplo, a comutatividade de quantificadores universais sucessivos. Embora o algoritmo de Hobbs e Shieber resulte em uma melhora significativa sobre o algoritmo ingênuo, ele ainda não resolve todo o problema de escopos de quantificadores, como observam os próprios autores [Hobbs-87]:

...precisam ser desenvolvidos métodos que usem conhecimento do mundo e considerações pragmáticas — raciocinando particularmente sobre quantidades e dependências entre entidades — para resolver a ambigüidade de escopo de quantificadores; e esses métodos necessitam ser integrados com o processamento sintático, semântico e pragmático requeridos na interpretação de linguagem natural.

Mostramos a seguir dois exemplos de sentenças que demonstram a eficiência da aplicação do algoritmo de Hobbs e Shieber.

Se na sentença:

Todo representante de uma companhia viu muitos exemplos

o sintagma preposicional de uma companhia fosse analisado como uma cláusula relativa restritiva reduzida — subclasse dos adjuntos adnominais —, sendo tal cláusula traduzida igualmente como cláusulas relativas restritivas, e se ela fosse aplicada a um algoritmo ingênuo, teríamos seis leituras para ela — referentes às 3! possíveis permutações de todo, uma e muitos. Entretanto, a seguinte ordem dos quantificadores todo, muitos, uma, fornece uma leitura ilegítima. Observe que esta ordem expressaria que, para cada representante, existe um grupo de muitos exemplos que ele viu; e, para cada exemplo que ele viu, existe uma companhia da qual ele é representante.

Para o entendimento da impossibilidade da ordem dos quantificadores *todo*, *muitos*, *uma*, Hobbs e Shieber usam a notação de Moore [Moore-86].

A notação de Moore diz que o conteúdo proposicional de uma sentença combina especificações que restringem o domínio de entidades quantificadas (*restrição*) com asserções sobre as entidades (*corpo*).

Como na ordem acima, o quantificador universal tem maior escopo, temos:

todo (*r*, *representante*(*r*) ... , ...)

O quantificador existencial fornece uma restrição ao domínio da variável *r* e deve portanto estar na restrição do universal:

todo (*r*, *representante*(*r*) & *uma* (*c*, *companhia*(*c*), *de* (*c*,*r*)), *viu*(*r*,*e*))

O quantificador *muitos* deve ter maior escopo no *corpo* do quantificador universal, mas também deve, por hipótese, ter maior escopo que o existencial na *restrição* do universal. Para ter maior escopo que ambos, ele deve ter maior escopo que o universal, mas isto viola a regra de escopo assumida (*todo*, *muitos*, *uma*).

Disto segue que, em geral, um determinado quantificador não pode vir depois do quantificador associado ao núcleo do sintagma nominal e antes do quantificador associado ao complemento do núcleo do sintagma nominal.

A redução no número de leituras válidas — neste exemplo de 6 para 5 — não foi representativa; mas, para a sentença:

Algum representante de todo departamento em muitas companhias
viu poucos exemplos de cada produto

onde seus três sintagmas preposicionais de todo departamento, em muitas companhias e de cada produto fossem analisados como cláusulas relativas restritivas reduzidas, existem somente 42 leituras válidas onde o algoritmo ingênuo geraria 120.

A descrição e a implementação Prolog do algoritmo de Hobbs e Shieber, bem como os formatos da linguagem de entrada e saída para o programa encontram-se no capítulo 7.

A comparação do algoritmo de Pereira — que gera todas as permutações de quantificadores — com o algoritmo de Hobbs e Shieber — que impede a geração de leituras ilegítimas, isto é, fórmulas lógicas mal formadas — também encontra-se no capítulo 7.

6.4 Tratamento de Quantificadores no Sistema TEAM

No Sistema TEAM a sentença em linguagem natural é primeiro transformada em uma forma lógica que por sua vez é transformada em uma *query language* que acessará a base de dados.

Portanto, a forma lógica possui um papel central por ser a mediadora entre:

- o modo que um usuário final vê as informações na base de dados (isto está relacionado com o tipo de perguntas que o usuário faz); e,
- o modo como a informação pode ser recuperada da base de dados através de perguntas expressas em uma *query language* para acesso a base de dados.

A forma lógica empregada neste sistema é baseada em uma lógica de primeira ordem estendida com alguns operadores — de ordem mais alta e outros intencionais — e aumentada com quantificadores para o artigo definido, pronomes interrogativos e para os números cardinais.

6.4.1 Escopo de Quantificadores

As regras de escopo não são aplicadas somente a quantificadores correspondentes a determinadores (cada, todo, algum, todos, os, etc.), mas também a operadores (negação, tempo, modo) e superlativos.

Para determinar o escopo apropriado, TEAM parte de informações da árvore sintática e associa dois tipos de informações adicionais a cada determinador: um *peso/poder* para o quantificador que ele introduz e um *quantificador associado* para a geração da forma lógica.

O peso associado a cada determinador permite a ele dominar outros determinadores. Por exemplo, o peso dado a *algum* é 5; a *o*, *um*, *todo* e *ninguém* é 10;

aos pronomes interrogativos *que/qual* e *nenhum* é 15; aos pronomes interrogativos *quem* e *o que* é 25 e à *any* é 35. Logo após comentaremos mais sobre este último determinador *que*, na língua portuguesa é traduzido como *qualquer*, *algum* ou *nenhum*.

Este peso associado se deve ao fato de que, embora as sentenças como:

Mostre o pico mais alto de cada país asiático.

Mostre o pico mais alto de um país asiático.

tenham a mesma estrutura sintática, elas requerem escopo relativo diferente para os quantificadores de país.

Como resposta da primeira sentença queremos uma lista de todos os países asiáticos com seus picos (*cada* governa o) e da segunda sentença queremos o pico mais alto da Ásia (*o* governa um).

Para gerar todos os escopos possíveis, TEAM usa um algoritmo do tipo geração-e-teste. Este algoritmo tenta gerar as $n!$ possíveis formas lógicas de uma sentença com n quantificadores, impondo algumas restrições para o número de possibilidades. Estas restrições são as seguintes:

- para uma sequência de variáveis quantificadas existencialmente, a rotina de geração produz somente um escopo, já que todas as permutações serão logicamente equivalentes. O mesmo ocorre para uma sequência quantificada universalmente; e,
- são eliminadas as possibilidades de geração de formas lógicas onde as variáveis aparecem fora do domínio de seu escopo.

Um filtro é aplicado para sentenças que possuem o determinador *any* (qualquer/algum/nenhum). Este filtro permite a TEAM manusear os vários comportamentos exibidos por *any*. O mapeamento de *any* para um quantificador do cálculo de predicados depende de seu contexto imediato. Por exemplo, se em uma sentença aparecer uma negação, ou, a sentença for interrogativa, o determinador *any* assume comportamentos diferentes, como ilustrado nas sentenças abaixo:

Eu posso escolher qualquer (*any*) homem no bar!

Eu posso escolher algum (*any*) homem no bar?

Se estas duas sentenças estivessem em inglês, levariam o *any* a ser mapeado para quantificador universal na primeira sentença e, na segunda, para o existencial. Como vimos acima, no português não temos este problema; pois, naturalmente usamos dois determinadores diferentes.

6.4.2 Críticas de Escopo

Como resultado do processo de geração temos um conjunto de formas lógicas alternativas, todas elas sendo interpretações logicamente possíveis. A essas formas lógicas é aplicado um conjunto de regras heurísticas (críticas) que lhes atribuem um determinado valor.

As críticas atualmente usadas por TEAM são heurísticas de disposição (ordem de entrada de determinadores na sentença) e heurísticas léxicas (escolha entre a leitura existencial ou universal para o determinador *any*, análise para os pares *qual/algum* e o peso/poder dos determinadores).

6.4.3 Informações Pragmáticas

Mesmo possuindo este mecanismo elaborado, o sistema TEAM falha para algumas sentenças no processo de geração do melhor escopo.

Por isso, informações pragmáticas específicas ao domínio/contexto podem ser essenciais para derivar a leitura correta dessas sentenças, podendo, inclusive, se sobrepor às críticas de escopo.

Como na sentença:

Qual é a altura de todos os picos no Nepal?

onde o sistema TEAM procura por uma única altura, que seria a altura de todos os picos no Nepal. Este escopo é correto para sentenças similares como:

Quem é o comandante de todas as tropas?

Mas, para a primeira sentença nenhuma pessoa produziria tal interpretação devido à informação pragmática de que cada pico tem sua própria altura. A aquisição e o uso de informações pragmáticas é um dos pontos fortes de TEAM.

6.4.4 Interação com o usuário

O sistema TEAM mostra ao usuário o resultado da aplicação das críticas de escopos; isto é, a forma lógica com o maior valor atribuído pelas regras de escopo.

A forma lógica sofre uma tradução para o inglês e é apresentada ao usuário para que ele avalie se a análise da sentença foi correta. Se o sistema TEAM for bem sucedido, esta tradução será uma paráfrase da sentença de entrada.

Capítulo 7

Avaliação de Dois Algoritmos Geradores de Escopos

Neste capítulo discutimos em detalhes dois algoritmos geradores de escopos — o algoritmo de Pereira [Pereira-86] e o algoritmo de Hobbs e Shieber [Hobbs-87].

Um gerador de escopo é um módulo que pode ser acoplado em um sistema de PLN à saída do módulo de Interpretação Semântica — como o de Pereira — ou, à saída do Analisador Sintático — como o de Hobbs e Shieber.

Para cada gerador analisado aqui, mostramos a sua implementação na linguagem Arity Prolog e algumas medidas a fim de avaliar a eficiência dos mesmos. A avaliação realizada refere-se somente à eficiência da tarefa de gerar escopos de quantificadores. Devemos observar que as medições de tempo de execução deram-se no Prolog interpretado e caso os programas geradores de escopo fossem compilados, o tempo obtido seria menor.

Discutimos também a inserção de um Módulo Pragmático e de um Módulo de Clarificação de Diálogos em um sistema de PLN. Isto para que, tal sistema assim formado — Analisadores Sintático e Semântico, Gerador de Escopos, Módulo Pragmático para escolha da interpretação correta da sentença e Módulo de Clarificação de Diálogo — consiga traduzir a sentença de entrada de acordo com a expectativa do usuário.

7.1 Gerador de Escopos de Pereira

7.1.1 Interpretação de Sentenças

Para a representação do significado das sentenças Pereira utiliza a Lógica de Primeira Ordem.

A Lógica Clássica não consegue representar os artigos definidos (o(a),os(as)), e, foi por isso que Colmerauer propôs um Sistema Trivalente com os valores de verdade verdadeiro, falso e indefinido. Este último é usado para representar situações onde o uso de artigos definidos deixa a sentença sem sentido. Vejamos a sentença abaixo:

O gato que Maria está segurando é preto.

Em situações onde Maria segura um gato, a sentença é verdadeira ou falsa dependendo de tal gato ser preto ou não. Em uma situação onde Maria está segurando mais de um gato, a sentença não é verdadeira nem falsa.

A gramática de Pereira, cuja listagem completa encontra-se no apêndice A, foi estendida e adaptada para o Português. Ela é restrita aos exemplos utilizados neste capítulo e deve ser ampliada se for necessário analisar novas sentenças. Os exemplos utilizados encontram-se no item 7.3.

Nela, por simplificação, ignoramos a pressuposição da unicidade representada pelo artigo definido singular o(a). Este artigo será tratado como o indefinido singular. Igual simplificação ocorre com o artigo definido plural.

Durante a execução por Prolog desta gramática, definida na notação de Cláusulas Definidas (GCD), o sintagma nominal $\text{ sint_nom(IV,S,G,N)$ é executado primeiro, embora seu argumento IV só será totalmente determinado quando o sintagma verbal $\text{ sint_ver(IV,N)$ for executado. Este tipo de operação mostra como a variável lógica ajuda unir uma expressão complexa sem ter que conhecer de antemão a especificação total de suas partes.

O programa de Pereira é ativado através do predicado $\text{ s/3$ mostrado abaixo:

```
s(T) --> sint_nom(IV,S,G,N),sint_ver(IV,N),
          terminador,{remove(S,T)}.
```

Os argumentos são definidos no item 7.1.2 enquanto que a descrição do gerador de escopos `remove/2` encontra-se no item 7.1.3.

A maioria das expressões intermediárias do significado são expressões λ , isto é, $\lambda X.f$. Elas são representadas em Prolog por termos da forma X^f . A GCD representa as expressões λ por variáveis lógicas e trata essas expressões λ de um modo puramente sintático.

Do ponto de vista de Prolog, o operador $^$ (operador existencial) representa um par de uma variável que será mais tarde amarrado a um quantificador ou trocado por uma constante e um fragmento de forma lógica contendo tal variável.

Veremos, abaixo, como interagir com o Interpretador Semântico-Gerador de Escopos de Pereira.

7.1.2 Linguagem de Entrada e de Saída

Ao programa de Pereira, adaptado para a gramática portuguesa, acoplamos o predicado `read_in/1` para que pudessemos digitar a sentença de entrada em linguagem natural. Este predicado transforma a sentença em uma lista de palavras, convertendo letras maiúsculas para minúsculas.

A linguagem de saída é uma estrutura Prolog envolvendo os operadores `&`, `=>` e `nao` (conectivos lógicos); os quantificadores universal `todo(x,p)` e existencial `existe(x,p)`. Por exemplo, a sentença de entrada:

Um rio atravessa cada país europeu.

através do programa de Pereira, é transformada em:

```
existe(_83E5,rio(_83E5) & todo(_8841,pais(_8841) & europeu(_8841)
=> atravessar(_83E5,_8841)))
```

```
todo(_8841,pais(_8841) & europeu(_8841) =>
existe(_83E5,rio(_83E5) & atravessar(_83E5,_8841)))
```

Figure 1

O método utilizado por Pereira para a geração de escopos baseia-se na construção de uma representação intermediária (árvore quantificada) cujos nós não terminais correspondem ao significado dos determinadores (nós quantificados) ou conectivos lógicos (nós conectivos). Os nós terminais (nós predicação) correspondem a traduções de substantivos comuns, verbos e outras palavras da sentença de entrada.

No programa, o nó quantificado é representado por uma estrutura da forma:

$$q(\text{De}, \text{Do}, \text{E})$$

onde De é o resultado do significado do determinante e Do e E são sub-árvores das quais o domínio e o escopo de De são obtidos. Um nó conectivo é representado pelo próprio conectivo aplicado aos seus argumentos.

Os nós predicação são representados por:

... P

Temos abaixo a árvore quantificada gerada pela GCD, para a sentença:

Um rio atravessa cada país europeu.

O predicado `remove/2` é o responsável pela tradução entre árvores quantificadas e fórmulas da Lógica de Primeira Ordem (LPO). Ele é ativado na cláusula `s/3`, mostrada abaixo:

```
s(T) --> sint_nom(IV,S,G,N),sint_ver(IV,N),terminador,
         {remove(S,T)}.
```

Para a definição dos argumentos dos predicados Prolog será usada a notação:

[A] < argi >, onde A é E/S, significando argumento de entrada ou saída, respectivamente ou ainda V quando o argumento é de entrada e saída; e, argi é o i-ésimo argumento.

Os argumentos de `remove/2` são os seguintes:

[E] < arg1 >: árvore quantificada da forma $q(De, Do, E)$ onde De é uma estrutura que representa o significado do determinador, Do e E são sub-árvores das quais o domínio e o escopo de De são obtidos.

Para a descrição do algoritmo usaremos a sentença Um rio atravessa cada país europeu; para ela este argumento é o seguinte:

```
q(_8421 ^ _842D ^ existe(_83F9,_8421 & _842D),... rio(_83F9),
  q(_89B9 ^ _89C5 ^ todo(_8855,_89B9 => _89C5),
    ... pais(_8855) & ... europeu(_8855),
    ... atravessar(_83F9,_8855)))
```

[S] < arg2 >: fórmula bem formada (fbf) da Lógica de Primeira Ordem. Este argumento, depois da execução de `remove/2` retornará por backtracking as duas fórmulas em LPO abaixo:

```
existe(_83E5,rio(_83E5) & todo(_8841,pais(_8841) & europeu(_8841)
=> atravessar(_83E5,_8841)))
```

```
todo(_8841,pais(_8841) & europeu(_8841) =>
existe(_83E5,rio(_83E5) & atravessar(_83E5,_8841)))
```

O predicado `remove/2`, mostrado abaixo, é formado por uma única cláusula com duas metas: `remove/3` e `aplicar_quants/3`.

```
remover(QuantTree,Formula) :- remover(QuantTree,Matrix,Store),
    aplicar_quants(Store,Matrix,Formula).
```

A primeira meta é o predicado `remover/3` com os argumentos:

[E] < arg1 > : árvore quantificada.

[S] < arg2 > : uma fórmula com variáveis livres.

[S] < arg3 > : uma lista de quantificadores cujos domínios foram determinados, mas não seus escopos.

A segunda meta é o predicado `aplicar_quants/3` com os argumentos:

[E] < arg1 > : a lista de quantificadores obtida de `remover/3`.

[E] < arg2 > : a fórmula com variáveis livres obtida de `remover/3`.

[S] < arg3 > : a fórmula gerada a partir da aplicação dos quantificadores do primeiro argumento na fórmula do segundo argumento.

Para a nossa sentença exemplo, os argumentos de `aplicar_quants/3` são:

< arg1 > :

```
[_8419 ^ existe(_83E5,rio(_83E5) & _8419),
 _89B1 ^ todo(_8841,pais(_8841) & europeu(_8841) => _89B1)]
```

< arg2 > :

```
atravessar(_83E5,_8841)
```

< arg3 > : fórmula bem formada da LPO mostrada acima como segundo argumento de `remover/2`.

O nome `remover` sugere a idéia de remover quantificadores para fora da lista — primeiro argumento de `aplicar_quants/3` — e aplicá-los a fórmula com variáveis livres — segundo argumento de `aplicar_quants/3` — para produzir uma fórmula fechada que é obtida através do terceiro argumento de `aplicar_quants/3`.

A ordem dos quantificadores no segundo argumento de `aplicar_quants/3` indica seus escopos relativos no resultado final; os quantificadores que aparecem primeiro na lista são escolhidos para ter maior escopo. Assim, a primeira fórmula fechada

em LPO terá o determinador um dominando o determinador cada. Quando ocorrer o backtracking, o primeiro argumento de aplicar_quants/3 será:

```
[_89B1 ^ todo(_8841,pais(_8841) & europeu(_8841) => _89B1),  
_8419 ^ existe(_83E5,rio(_83E5) & _8419)]
```

e a fórmula da LPO resultante terá o quantificador cada dominando o determinador um.

Vamos detalhar agora o predicado `remove/3`. A primeira cláusula de `remove/3` é:

```
remove(...Predication,Predication,[]).
```

sendo seus argumentos os seguintes:

[E] < arg1 > : nó predicação, isto é, as traduções dos substantivos comuns, verbos, adjetivos, etc.

[S] < arg2 > : fórmula gerada para o nó predicação.

[S] < arg3 > : lista de quantificadores, que para esses nós é vazia.

Esta cláusula fará com que:

...rio(_83E5) se transforme em rio(_83E5);

...pais(_8841) se transforme em pais(_8841);

...europeu(_8841) se transforme em europeu(_8841); e

...atravessar(_83E5,_8841) se transforme em atravessar(_83E5,_8841).

quando esta cláusula for chamada dentro das duas outras cláusulas de `remove/3`.

A segunda cláusula de `remove/3` é a seguinte:

```
remove(QuantTree1 & QuantTree2, Formula1 & Formula2,Store) :-  
    remove(QuantTree1,Matrix1,Store1),
```

```

remover(QuantTree2,Matrix2,Store2),
concatenar(Pass1,Apply1,Store1),
concatenar(Pass2,Apply2,Store2),
aplicar_quants(Apply1,Matrix1,Formula1),
aplicar_quants(Apply2,Matrix2,Formula2),
misturar(Pass1,Pass2,Store).

```

Esta cláusula trata os nós conectivos (&) que surgem das traduções de adjetivos e cláusulas relativas restritivas. Seus argumentos são os seguintes:

[E] < arg1 > : conectivo & infixo, seus argumentos são duas árvores quantificadas.

[S] < arg2 > : conectivo & infixo, seus argumentos são duas fórmulas com variáveis livres.

[S] < arg3 > : lista de quantificadores de toda a conjunção.

Para a nossa sentença exemplo, esta cláusula é chamada com o seu primeiro argumento sendo:

```
...pais(_8841) & ...europeu(_8841)
```

sendo os seus argumentos de saída os seguintes:

< arg2 > :

```
pais(_8841) & europeu(_8841)
```

< arg3 > : []; pois não existem quantificadores no argumento de entrada.

Obtivemos esta saída pois:

a primeira meta da segunda cláusula de `remover/3` devolveu nos seus segundo e terceiro argumentos de saída, `pais(_8841)` e `[]` respectivamente.

a segunda meta devolveu nos seus segundo e terceiro argumentos de saída, `europeu(_8841)` e `[]` respectivamente.

a terceira e quarta metas consistem do predicado `concatenar/3` — toma duas listas e retorna uma nova lista tendo como resultado a concatenação das duas primeiras. Como usamos `concatenar/3` para quebrar uma lista em duas outras, para o nosso

exemplo, os dois primeiros argumentos das metas `concatenar/3` também retornam listas vazias.

a quinta e sexta metas, para o nosso exemplo, unificam o segundo argumento com o terceiro.

a sétima meta é `misturar/3` — toma duas listas, l e r , produzindo uma terceira lista $s = l_1.r_1, \dots, l_k.r_k$ sendo $.$ a concatenação de duas listas resultando uma terceira. As listas $l_1 \dots l_k$ e $r_1 \dots r_k$ são tal que $l = l_1 \dots l_k$ e $r = r_1 \dots r_k$. Assim, `misturar/3` embaralha os elementos de l e r mas mantém a ordem original de seus elementos.

No nosso exemplo, `misturar/3` toma em seus dois primeiros argumentos listas vazias retornando em seu terceiro argumento uma lista vazia.

A segunda cláusula de `remove/3` pode ser resumida assim:

1. obtenção da fórmula e lista de quantificadores para cada lado da conjunção — feito pelas duas metas `remove/3`;
2. uso da meta `concatenar/3` para quebrar cada lista de quantificadores em uma lista de nome `Passi` e outra de nome `Applyi`, sendo i igual a 1 quando `concatenar/3` trata da lista associada com a árvore quantificada esquerda e i é igual a 2 para a árvore direita;
3. cada lista `Applyi` é aplicada a sua correspondente fórmula com variáveis livres através de `aplicar_quants/3`; e
4. as listas `Passi` são misturadas para formar o terceiro argumento da segunda cláusula de `remove/3`.

A terceira cláusula de `remove/3` trata o nó quantificado $q(De, Do, E)$:

```
remove(q(Range^Quant, RangeTree, ScopeTree), Matrix, Store) :-
    remove(RangeTree, RangeMatrix, RangeStore),
    remove(ScopeTree, Matrix, ScopeStore),
    concatenar(RangePass, RangeApply, RangeStore),
    aplicar_quants(RangeApply, RangeMatrix, Range),
    concatenar(RangePass, [Quant], Pass),
    misturar(Pass, ScopeStore, Store).
```


Das três cláusulas `remove/3` esta é a primeira cláusula chamada quando temos sentenças com quantificadores. Seus argumentos são os seguintes:

[E] < arg1 > : árvore quantificada, sendo distinguidas as subárvores `Do` e `E` que fornecerão o domínio e o escopo de `De`.

[S] < arg2 > : fórmula com variáveis livres.

[S] < arg3 > : lista de quantificadores.

Na nossa sentença `Um rio atravessa cada país europeu`, a árvore quantificada é a seguinte:

```
q(_8421 ^ _842D ^ existe(_83F9, _8421 & _842D), ... rio(_83F9),
  q(_89B9 ^ _89C5 ^ todo(_8855, _89B9 => _89C5),
    ... pais(_8855) & ... europeu(_8855), ... atravessar(_83F9, _8855)))
```

onde:

`Quant` é instanciada com

```
_842D ^ existe(_83F9, _8421 & _842D)
```

`RangeTree` é instanciada com

```
... rio(_83F9)
```

`ScopeTree` é instanciada com

```
q(_89B9 ^ _89C5 ^ todo(_8855, _89B9 => _89C5),
  ... pais(_8855) & ... europeu(_8855), ... atravessar(_83F9, _8855))
```

As chamadas recursivas a `remove/3` devolvem a fórmula com variáveis livres e a lista de quantificadores para `RangeTree` e `ScopeTree`. Para `RangeTree` — `Do` — a fórmula é `rio(_83F9)` e a lista é vazia. Para `ScopeTree` — `E` — a fórmula é `atravessar(_83F9, _8855)` e a lista é:

```
[_89B1 ^ todo(_8841,pais(_8841) & europeu(_8841) => _89B1)]
```

A lista do domínio — Do — é dividida em duas outras RangePass e RangeApply. Os quantificadores de RangeApply são aplicados à fórmula obtida de remover/3 relacionada com o domínio, para obtermos o domínio Range que no nosso exemplo é rio(_83F9).

A lista de quantificadores RangePass, através de concatenar/3, é concatenada com a lista [Quant]; no nosso exemplo, a saída de concatenar/3 é a própria [Quant].

O resultado de concatenar/3 é misturado com a lista obtida do escopo fornecida pela segunda meta, para assim formar a lista de quantificadores do nó inteiro, que para o nosso exemplo é a seguinte:

```
[_8419 ^ existe(_83E5,rio(_83E5) & _8419),  
_89B1 ^ todo(_8841,pais(_8841) & europeu(_8841) => _89B1)]
```

e quando tivermos o backtracking será:

```
[_89B1 ^ todo(_8841,pais(_8841) & europeu(_8841) => _89B1),  
_8419 ^ existe(_83E5,rio(_83E5) & _8419)]
```

O não determinismo na produção de escopos alternativos surge do uso dos predicados misturar/3 e concatenar/3 que operam sobre listas e são mostrados abaixo:

```
misturar([],[],[]).
```

```
misturar([Elem|LeftList],RightList,[Elem|List]) :-  
    misturar(LeftList,RightList,List).
```

```
misturar(LeftList,[Elem|RightList],[Elem|List]) :-  
    misturar(LeftList,RightList,List).
```

```
concatenar([],List,List).
```

`concatenar([Elem|Front],Back,[Elem|List]) :-
concatenar(Front,Back,List).`

7.2 Gerador de Escopos de Hobbs e Shieber

O algoritmo de Hobbs e Shieber está baseado na operação primitiva de aplicar um termo complexo (a definição encontra-se no item 7.2.1) na fórmula bem formada (fbf) em que ele ocorre. Um termo complexo em uma fórmula bem formada é trocado pela variável que ele restringe; e, tal variável está desta forma amarrada por conter a fórmula completa no quantificador apropriado.

Assim, aplicar o termo $\langle q \ x \ r(x) \rangle$ a uma fbf contendo tal termo complexo, isto é, $p(\langle q \ x \ r(x) \rangle)$, produz a fbf quantificada $q(\langle x, r(x), p(x) \rangle)$. Esta é a operação primitiva pela qual termos complexos são removidos de uma fbf e fbf quantificadas são introduzidas.

A listagem completa do algoritmo de Hobbs e Shieber encontra-se no apêndice A.

7.2.1 Linguagem de Entrada e de Saída

A sentença abaixo é usada para ilustrar a forma de entrada e saída do algoritmo de Hobbs e Shieber.

Um representante de cada departamento em muitas companhias
analisou todos os processos.

Como linguagem de entrada para o algoritmo, devemos fornecer a sua estrutura predicado-argumentos, como é mostrada abaixo sob uma notação simplificada:

```

analizou(<um r
  e( representante(r),
    de(r,<cada d
      e( departamento(d),
        em(d,<muitas c companhia(c)>>>>>>)),
    <todos os p processo(p)>>)

```

É responsabilidade do analisador sintático a geração desta linguagem de entrada, que torna explícita as relações predicado-argumentos e as relações de subordinação gramatical.

Uma fbf desta linguagem de entrada consiste de um predicado (verbo da sentença) ou um operador opaco aplicado a um ou mais argumentos. Um argumento pode ser uma constante, uma variável, outra fbf, ou o que é chamado de um termo complexo. Um termo complexo é uma tripla ordenada consistindo de um quantificador, uma variável e uma fbf (chamada de restrição), que representa a predicação que é gramaticalmente subordinada à variável.

Na linguagem de entrada mostrada acima, um termo complexo pode ser lido como: quantificador variável tal que restrição, por exemplo, muitas c tal que c é uma companhia — < muitas c companhia(c) >.

A linguagem de saída é idêntica à de entrada, exceto que não contem termos complexos. Os quantificadores são expressados na linguagem de saída como operadores com três argumentos — a variável ligada ao quantificador, uma fbf restringindo o domínio da variável quantificada e o corpo alcançado pela quantificação, isto é:

quantificador(Variável, Restrição, Corpo)

Uma das 14 saídas que o algoritmo de Hobbs e Shieber fornece — mostrada aqui sob uma notação simplificada — para a sentença:

Um representante de cada departamento em muitas companhias
analizou todos os processos.

é a seguinte:

```

muitas(c,companhia(c),
      todos os(p,processo(p),
               um(r, cada(d, e(departamento(d),em(d,c)),
                               e(representante(r),de(r,d)))),
               analisar(r,p))))

```

7.2.2 Escopo de Predicados Opacos

Além de tratar do escopo de quantificadores relativo a outros quantificadores, o algoritmo de Hobbs e Shieber também permite que quantificadores sejam escopados dentro ou fora de predicados opacos.

Predicados opacos/intencionais são aqueles em que ao menos um de seus argumentos é uma fbf.

Como exemplo de linguagem de entrada com um predicado opaco, temos:

```
fbf(nao,[fbf(e_preto,[termo(todo,x,[fbf(pato,[x]))])])])])
```

que é a estrutura predicado-argumentos para a sentença:

Todo pato não é preto.

Devemos declarar que `not` é um operador opaco em seu único argumento, dessa forma:

`opaco(nao,1)`

onde o primeiro argumento de `opaco` é o nome do predicado e o segundo é o índice da posição na qual o predicado é opaco.

A saída do algoritmo corresponde aos dois escopos relativos do quantificador universal e da negação:

```
fbf(todo,[x,[fbf(pato,[x])],fbf(nao,[fbf(e_preto,[x])])])
```

```
fbf(nao,[fbf(todo,[x,[fbf(pato,[x])],fbf(e_preto,[x])])])])
```

7.2.3 Descrição do Algoritmo

O funcionamento da implementação Prolog do algoritmo de Hobbs e Shieber é demonstrado através do uso da sentença:

Um representante de cada departamento em muitas companhias
analisou todos os processos.

Como dito no item 7.1, a linguagem de entrada para o algoritmo é a sua estrutura predicado-argumentos:

$$p(\text{arg1}, \text{arg2}, \dots, \text{argn})$$

que é representada pelo termo Prolog:

$$\text{fbf}(p, [\text{arg1}', \text{arg2}', \dots, \text{argn}'])$$

com argi' sendo a codificação da subexpressão argi . Se argi é uma constante ela é representada por uma constante Prolog de mesmo nome; se é um termo complexo é representada pelo termo Prolog:

$$\text{termo}(\text{Quant}, \text{Var}, \text{Restrição}')$$

onde $\text{Restrição}'$ é a codificação da fbf que forma a restrição do quantificador.

Assim, para a sentença acima, o programa de Hobbs e Shieber é ativado através do predicado `gerar/2`:

$$\text{gerar}(\text{Form}, \text{ScopedForm})$$

com seus argumentos sendo:

[E] < arg1 >: estrutura predicado-argumentos da sentença.

Para a nossa sentença esse argumento é:

```

1) fbf(analisar,[termo(um,r,fbf(and,
                        [fbf(representante,[r]),
                          fbf(de,[r,termo(cada,d,
                                fbf(and,
                                  [fbf(departamento,[d]),
                                    fbf(em,[d,termo(muitas,c,
                                                fbf(companhia,[c]))]))]))]),
                        termo('todos os',p,fbf(processo,[p]))))

```

[S] < arg2 >: todos os escopos permitidos para a sentença acima.

O predicado gerar/2 é formado por uma única cláusula que possui a meta remover/3:

```

gerar(Form,ScopedForm) :-
    remover(Form,true,ScopedForm).

```

A meta remover/3, mostrada abaixo, consiste de uma única cláusula que possui duas metas: remover_args_opacos/2 e aplicar_termos/3.

```

remover(Form,Complete,ScopedForm) :-
    remover_args_opacos(Form,PulledOpaque),
    aplicar_termos(PulledOpaque,Complete,ScopedForm).

```

Os argumentos de remover/3 são os seguintes:

[E] < arg1 >: representação da sentença de entrada.

[E] < arg2 >: o valor true para que retornem somente expressões sem nenhum termo complexo, expressões estas, chamadas de totalmente escopadas (outro tipo são as parcialmente escopadas que serão vistas logo mais).

[S] < arg3 >: todos os escopos permitidos para a sentença.

O predicado remover_args_opacos/2 é formado pelas duas cláusulas abaixo:

```

remover_args_opacos(fbf(Pred,Args),fbf(Pred,ScopedArgs)) :- !,
    remover_args_opacos(Pred, 1, Args,ScopedArgs).

```

```

remover_args_opacos(Term,Term).

```

e seus argumentos são os seguintes:

[E] < arg1 >: um termo complexo ou uma fbf com termos complexos.

[S] < arg2 >: o próprio < arg1 > se em < arg1 > tivermos um termo; se tivermos uma fbf é a fbf com os argumentos opacos recursivamente escopados.

No exemplo, a chamada a `remover_args_opacos/2` retorna no segundo argumento a fbf original desde que o primeiro argumento não possui nenhum operador opaco.

A primeira cláusula de `remover_args_opacos/2` gera uma fbf total ou parcialmente escopada; é formada pela meta `remover_args_opacos/4`.

A meta `remover_args_opacos/4` contem as três cláusulas abaixo:

```
% Fim dos argumentos
remover_args_opacos(_, _, [], []) :- !.

% Argumento corrente e' opaco
remover_args_opacos(Pred, ArgIndex,
    [FirstArg|RestArgs], [ScopedFirstArg|ScopedRestArgs]) :-
    opaco(Pred, ArgIndex), !,
    remover(FirstArg, false, ScopedFirstArg),
    NextIndex is ArgIndex+1,
    remover_args_opacos(Pred, NextIndex, RestArgs, ScopedRestArgs).

% Argumento corrente nao e' opaco
remover_args_opacos(Pred, ArgIndex,
    [FirstArg|RestArgs], [ScopedFirstArg|ScopedRestArgs]) :-
    remover_args_opacos(FirstArg, ScopedFirstArg),
    NextIndex is ArgIndex+1,
    remover_args_opacos(Pred, NextIndex, RestArgs, ScopedRestArgs).
```

com seus argumentos sendo:

[E] < arg1 >: o predicado da fbf cujos argumentos estão sendo escopados.

[E] < arg2 >: o n-ésimo índice do argumento correntemente sendo escopado.

[E] < arg3 >: lista de argumentos indexadas pelo segundo argumento de `remover_args_opacos/2`.

[S] < arg4 >: lista com posições opacas recursivamente escapadas.

Por exemplo, se *p* for um predicado opaco em seu único argumento, então, para a fbf:

$$p(s(<q \ x \ r(x)>))$$

`remover_args_opacos/4` gera a fbf:

$$p(q(x,r(x),s(x)))$$

ou a inalterada fbf:

$$p(s(q \ x \ r(x)))$$

No primeiro caso, o predicado opaco *p* tem escopo maior que o quantificador *q*; no último, o quantificador *q* não foi aplicado ainda e a fbf irá subsequentemente produzir leituras onde *q* terá escopo maior que *p*.

Continuando a análise do predicado `remover/3`, temos como sua segunda meta o predicado `aplicar_termos/3` que é composto de três cláusulas:

```
aplicar_termos(Form,_,Form) :-  
    not(termo(Form,_)), !.
```

```
aplicar_termos(Form,false,Form).
```

```
aplicar_termos(Form,Complete,ScopedForm) :-  
    termo_aplicavel(Form,Term),  
    aplicar(Term,Form,AppliedForm),  
    aplicar_termos(AppliedForm,Complete,ScopedForm).
```

No nosso exemplo, a cláusula chamada é a terceira desde que a primeira cláusula só é verdadeira quando não existem mais termos complexos no primeiro argumento enquanto que a segunda só é verdadeira quando seu segundo argumento for igual a `false`.

A chamada a `termo_aplicavel/2` retorna uma lista de todos os termos complexos disjuntos. Para o exemplo mostrado em 1) são dois:

```

2) termo(um,r,fbf(and,
                [fbf(representante,[r]),
                 fbf(de,[r,termo(cada,d,
                                fbf(and,
                                    [fbf(departamento,[d]),
                                     fbf(em,[d,termo(muitas,c,
                                                    fbf(companhia,[c]))]))]))]))))

```

```

3) termo('todos os',p,fbf(processo,[p]))

```

O predicado `aplicar/3` é chamado para cada um destes termos complexos; sendo seus argumentos os seguintes:

[E] < arg1 >: um termo complexo.

[E] < arg2 >: uma fbf contendo tal termo complexo, ela possui a forma: $p(\text{termo}(q,x,r(x)))$.

[S] < arg3 >: uma fórmula bem formada quantificada da forma: $q(x,r(x),p(x))$.

O predicado `aplicar/3` é o seguinte:

```

aplicar(termo(Quant,Var,Restrict),Body,
        fbf(Quant,[Var,PulledRestrict,OutBody])) :-
    remover(Restrict,false,PulledRestrict),
    subst(Var,termo(Quant,Var,Restrict),Body,OutBody).

```

Através de `aplicar/3` termos complexos são removidos de uma fbf e fbf quantificadas são geradas. Dentro dele existe uma chamada recursiva a `remover/3` sobre a restrição do termo complexo que gera todos os escopos possíveis para a restrição.

Quando `aplicar/3` é chamado com seu primeiro argumento unificado com 1) e seu primeiro argumento unificado com 2), o resultado dos escopos da restrição de 2) são as quatro fbf apresentadas aqui sob a notação simplificada utilizada no item 7.2.1:

- 4) `and(representante(r),
 de(r,<cada d
 and(departamento(d),
 em(d,<muitas c companhia(c)>>>)))`
- 5) `cada(d, and(departamento(d), em(d, <muitas c companhia(c)>)),
 and(representante(r), de(r, d)))`
- 6) `muitas(c, companhia(c), cada(d, and(departamento(d), em(d, c)),
 and(representante(r), de(r, d))))`
- 7) `cada(d, muitas(c, companhia(c), and(departamento(d), em(d, c))),
 and(representante(r), de(r, d)))`

Pelo fato desta chamada a `remover/3` ter o segundo argumento instanciado com `false`, teremos a própria restrição como retorno (4) bem como a fbf parcialmente escopada (5) juntamente com as fbf totalmente escopadas da restrição (6) e (7).

A fbf (4) gera as duas leituras em que as variáveis `d` e `c` tem maior escopo que `r`. A fbf (5) é também parcial pois possui um termo complexo; ela produz uma leitura em que `r` tem maior escopo que `d` mas não que `c`; o termo complexo para `c` espera uma aplicação que dará maior escopo a ele. As fbf (6) e (7) produzem leituras em que `d` e `c` estarão dentro do escopo de `r`.

Cada uma destas fbf se tornarão a restrição em uma fbf quantificada construída por `aplicar/3`. Assim, da restrição 5), `aplicar/3` construirá a seguinte fbf quantificada, também em notação simplificada:

- 8) `um(r, cada(d, and(departamento(d),
 em(d, <muitas c companhia(c)>>)),
 and(representante(r), de(r, d))),
 analisar(r, <todos os p processos(p)>))`

A recursão de cauda de `aplicar_termos/3` torna os termos complexos remanescentes — `< muitas c companhia(c)>` e `< todos os p processos(p)>` — quantificadores de largo escopo.

Assim, em 8) `c` e `p` ganharão maior escopo que `r` e `d`. Finalmente, uma das leituras geradas a partir de 8) é:

```

fbf(muitas,[c,fbf(companhia,[c]),
    fbf(todos os,[p,fbf(processos,[p]),
        fbf(cada,[d,fbf(and,[fbf(departamento,[d]),
            fbf(em,[d,c]))]),
        fbf(um,[r,fbf(and,
            [fbf(representante,[r]),fbf(de,[r,d]))]),
            fbf(analisar,[r,a]))]))]))))

```

7.3 Tempo de Execução e Avaliação

Para comparar a eficiência dos dois algoritmos, foram realizadas várias medidas referentes a tempo de execução (tempo de CPU) e número de interpretações geradas para as seguintes sentenças:

- 1) Todos os estudantes usam um microcomputador.
- 2) Todos os estudantes usam um vax.
- 3) Um rio atravessa cada país europeu.
- 4) Cada presidente de um país sul americano visitou Brasília.
- 5) A capital de cada país europeu festejou a Revolução Francesa.
- 6) A altura de todos os picos europeus ultrapassa mil metros.
- 7) Os guerreiros têm uma lanca.
- 8) Os guerreiros embarcaram em um navio.
- 9) Todo cachorro persegue um gato.
- 10) Todo vigia de uma companhia dorme.
- 11) Cada representante de uma companhia trouxe uma amostra grátis.
- 12) O comandante de todas as tropas visitou o papa.
- 13) Cada policial de São Carlos persegue o bandido que assaltou um banco.
- 14) Nenhum guerreiro que embarcou em um navio voltou para a terra natal.
- 15) Um representante de cada departamento analisou todos os processos de

Sentença	Pereira		Hobbs– Shieber	
	Nº Traduções	Tempo Execução	Nº Traduções	Tempo Execução
1	2	4.2	2	90.4
2	2	4.2	2	90.5
3	2	5.2	2	126.2
4	2	5.8	1	62.1
5	2	5.8	1	62.2
6	2	5.9	1	62.2
7	2	4.1	2	90.4
8	2	4.1	2	90.4
9	2	4.1	2	90.4
10	2	4.7	1	51.7
11	5	10.8	2	376.9
12	5	9.3	2	308.7
13	7	15.5	5	602.5
14	7	17.2	5	587.4
15	14	23.7	2	802.8
16	26	52.4	14	2411.7

Tabela 7.1: Pereira *versus* Hobbs–Shieber: Nº Traduções e Tempo Execução

cada produto

16) Os guerreiros que embarcaram em um navio têm uma esposa bonita que espera o regresso.

As medições do tempo de execução foram realizadas a partir de uma representação que serve como entrada para os dois algoritmos geradores. No algoritmo de Pereira tal representação é a árvore quantificada enquanto que no de Hobbs e Shieber é a estrutura predicado-argumentos da sentença.

O tempo de execução dos algoritmos estão em centésimos de segundos e foram obtidos usando-se um microcomputador PC-AT compatível.

A tabela 7.1 mostra a comparação entre os dois algoritmos. É interessante destacar que o tempo médio para realizar cada tradução para o algoritmo de Pereira é 2.3 centésimos de segundos, com um pequeno desvio, ou seja, é praticamente constante.

A primeira impressão sobre o tempo de execução do algoritmo de Hobbs e Shieber leva a pensar que um algoritmo que suprime leituras que são falsas devido

Sentença	Tempo Execução		Tempo (H-Sh/Pe) em Pe-H-Sh Traduções					
	Pereira	Hobbs-Shieber	2-2	2-1	5-2	7-5	14-7	26-14
1	4.2	90.4	21.5					
2	4.2	90.5	21.5					
3	5.2	126.2	24.3					
4	5.8	62.1		10.7				
5	5.8	62.2		10.7				
6	5.9	62.2		10.5				
7	4.1	90.4	22.0					
8	4.1	90.4	22.0					
9	4.1	90.4	22.0					
10	4.7	51.7		11.0				
11	10.8	376.9			34.9			
12	9.3	308.7			33.2			
13	15.5	602.5				38.9		
14	17.2	587.4				34.2		
15	23.7	802.8					33.9	
16	52.4	2411.7						46.0

Tabela 7.2: Tempo H-Sh/ Tempo Pe para Pe-H-Sh Traduções

a estruturação da sua fórmula lógica não seria adequado.

Na tabela 7.2 mostramos, para a mesma relação entre o número de sentenças geradas pelo algoritmo de Pereira e o número de sentenças geradas pelo algoritmo de Hobbs e Shieber, quantas vezes mais tempo, em relação a Pereira, o algoritmo de Hobbs e Shieber necessita para gerar as traduções.

Podemos observar que, no caso de ambos os algoritmos gerar o mesmo número de traduções — 2 para Pereira e 2 para Hobbs e Shieber —, o algoritmo de Hobbs e Shieber usa, aproximadamente, 22 vezes mais tempo que o de Pereira. No caso em que Pereira gera 2 traduções e Hobbs e Shieber 1, este número é aproximadamente 11, isto é, metade que o anterior.

Os casos que geram respectivamente 5-2, 7-5 e 14-7 traduções, parecem ser semelhantes, e, nestes casos, o algoritmo de Hobbs e Shieber usa, aproximadamente 35 vezes mais tempo que o de Pereira.

Portanto, em termos absolutos de tempo de execução, o algoritmo tipo *força bruta* de Pereira, ainda que sem a *inteligência* do algoritmo de Hobbs e Shieber que consegue suprimir fórmulas lógicas com leituras falsas, é claramente superior.

Nº de Determinadores	Pereira Nº Traduções	Hobbs-Shieber Nº Traduções
1	1	1
2	2	1-2
3	5-7	2-5
4	14-26	2-14

Tabela 7.3: Pereira *versus* Hobbs-Shieber: Geração aproximada do Nº de Traduções

Mas, um fato que deve ser considerado é que os geradores de escopos, após gerar cada tradução, devem acessar informações que levarão à escolha da interpretação mais plausível para a sentença.

Em sistemas que possuem geradores de escopo (o TEAM, por exemplo), ou no Sistema de Reescrita de Saint-Dizier (ambos vistos no capítulo 6), a ambigüidade è retirada através de:

- informações contextuais/conhecimento do mundo;
- paráfrases dos escopos alternativos, em alguma linguagem mais próxima possível da natural; e,
- críticas de escopo.

Portanto, uma redução no número de interpretações diminuirá o tempo de acesso das informações que levarão à escolha da interpretação desejada pelo usuário. Assim, o algoritmo de Pereira poderá ficar em desvantagem quando as suas $n!$ ou mais — traduções logicamente equivalentes — saídas tiverem que ser avaliadas.

Na tabela 7.3, mostramos uma comparação aproximada entre o número de determinadores da sentença de entrada e o número de interpretações geradas por cada algoritmo:

Uma observação que deve ser feita é sobre o fato do algoritmo de Pereira gerar somente cinco interpretações para as sentenças 11 e 12 enquanto gera sete para as sentenças 13 e 14, mesmo embora todas possuam três determinadores. Ocorre que, substantivos que possuem complementos nominais são traduzidos, na nossa GCD, através de um predicado binário. Por exemplo, uma fórmula lógica para a sentença:

O comandante de todas as tropas visitou o papa.

é a seguinte:

```
existe(_B809,papa(_B809) &
  todo(_B175,tropa(_B175) =>
    existe(_AE69,comandante(_AE69,_B175)
      & visitar(_AE69,_B809))))
```

Nesta fórmula lógica podemos ver que *comandante* é traduzido para o predicado binário *comandante(X,Y)* onde a variável no primeiro argumento diz respeito ao comandante em questão e, a variável no segundo argumento está relacionada com o sintagma preposicional *de todas as tropas*. Como o algoritmo de Pereira se propõe a gerar somente fórmulas bem formadas fechadas, essas traduções em predicados binários limitarão as possibilidades de combinação dos determinadores.

Isto não ocorre com cláusulas relativas pois suas traduções não agrupam variáveis em predicados binários e sim, as liga através da conjunção (&). Por exemplo, para a sentença:

Todo homem que ama uma mulher vive.

uma das traduções é a seguinte:

```
todo(H, homem(H) & existe(M, mulher(M) & ama(H,M)) => vive(H)).
```

Como podemos observar as variáveis *H* e *M* não estão ligadas na tradução do substantivo *homem*.

Semelhante diferenciação no tratamento de complementos nominais e cláusulas relativas ocorre na fórmula bem formada que serve como entrada para o algoritmo de Hobbs e Shieber.

7.4 Considerações: Módulo Pragmático e Módulo de Clarificação de Diálogos

Em sistemas de PLN, uma sentença de entrada pode ser sintática e semanticamente bem formada e, ainda assim, violar as propriedades estruturais do modelo de mundo do sistema. Isto não quer dizer que a pessoa que a digitou/pronunciou tenha necessariamente uma visão incorreta do mundo, ou, mesmo uma que difira da visão do sistema, mas somente que a representação semântica da sentença não está de acordo com o modelo de mundo do sistema. Estas sentenças são chamadas de *pragmaticamente mal formadas*.

Tomemos como exemplo a sentença:

Todos os estudantes usam um microcomputador.

Como saída de um gerador de escopos, devemos ter as duas interpretações possíveis para a sentença acima, aqui mostradas na figura 7.1 sob a forma de uma árvore quantificada de três ramos.

Mas, o conhecimento do mundo real nos diz que:

Um microcomputador serve a somente um estudante.

isto é, a relação entre as entidades estudante e microcomputador é de 1:1, como mostrada pela figura 7.2. Esta regra esta correta se tomamos um microcomputador como sendo um computador pessoal.

Portanto, como esta informação está de acordo com a interpretação da primeira árvore, podemos descartar a segunda. A segunda árvore nos diz que existe um único micro para todos os estudantes. Interpretações onde o quantificador um domina o quantificador todos os são corretas, por exemplo, para a sentença:

Todos os estudantes usam um Vax.

pois para tal sentença, temos a relação entre entidades mostrada pela figura 7.3, isto é, a relação entre estudante e Vax é de N:1 respectivamente, e é obtida através da informação pragmática:

Figura 7.1: Interpretações de: Todos os estudantes usam um microcomputador.

Figura 7.2: Relação: Estudante—Microcomputador.

Figura 7.3: Relação: Estudante—Vax.

Vários estudantes acessam um Vax, pois ele é compartilhado.

Assim, com o uso de conhecimento do mundo e informações contextuais que raciocinam sobre quantidades e relações entre as entidades do universo de discurso, podemos escolher a interpretação correta de uma sentença.

Outra função desejada para um Módulo Pragmático é que ele diminua o número de buscas na base de conhecimento de um sistema de PLN. Por exemplo, sabendo-se que a sentença interrogativa:

Qual é o comandante de todas as tropas?

possui um único nome como resposta, regras pragmáticas poderiam fornecer o nome procurado terminando a busca na base, logo após de encontrar a primeira cláusula que respondesse a pergunta.

Quando em um Módulo Pragmático não encontramos as regras que definem a interação entre as entidades, uma solução seria usar as várias interpretações do Módulo Gerador de Escopos e, depois de transformá-las para uma linguagem mais natural, apresentá-las ao usuário para que ele decida a interpretação correta de sua sentença. De acordo com nosso desejo de fornecer a interpretação correta de uma sentença, do ponto de vista da ordem dos quantificadores, este seria o papel do Módulo de Clarificação de Diálogos.

O sistema de PLN assim constituído, poderia inclusive aprender com a informação fornecida pelo usuário.

Um esquema de interação entre os Módulos Gerador de Escopos, Pragmático e de Clarificação de Diálogos poderia ser o mostrado pela figura 7.4.

Figura 7.4: Interação: Gerador de Escopos–Regras Pragmáticas–Clarificação de Diálogos.

Capítulo 8

Conclusão

No presente trabalho incluímos um capítulo dedicado a semântica devido a íntima relação entre a ambigüidade de escopo de quantificadores e o significado da sentença. Propusemos também uma nova classificação para semântica em PLN, tendo em vista os avanços que suscitaram nos sistemas de PLN a necessidade de outras abordagens para semântica. Como exemplo, podemos citar a Semântica de Situações e Baseadas em Conhecimento que estão sendo recentemente usadas em Processamento de Linguagem Natural — PLN.

Mesmo na abordagem de semântica dentro de Sistemas Lógicos, novas lógicas e, em especial, as lógicas não padrão, tais como lógicas não monotônicas, default, lógica nebulosa, estão sendo igualmente cogitadas.

Uma característica que se deve levar em consideração é a importância da computabilidade e o desempenho razoável em sistemas com essas novas abordagens.

Iniciamos um estudo de pragmática, pela sua importância no tratamento de ambigüidades em geral e, em especial, a de escopo de quantificadores — tema central deste trabalho.

Depois de uma análise feita sobre os trabalhos de vários autores apresentados nos capítulos 5 e 6, podemos dizer que o problema de selecionar o escopo mais plausível de quantificadores pode ser resolvido de duas maneiras:

1. pela aplicação repetida de algum conjunto de regras de escopos sobre uma representação intermediária do significado da sentença, de modo a produzir uma forma lógica final que satisfaça todas as regras de escopos relevantes.
2. pela geração e comparação de todos os escopos possíveis, ou relevantes.

Partimos assim, para a avaliação de dois algoritmos geradores de escopos por entendermos que tendo em mãos todos os escopos possíveis de uma sentença, não falharíamos na obtenção da interpretação correta. Os algoritmos são o de [Pereira-86] e [Hobbs-87].

Em nosso trabalho, o problema de se retirar a ambigüidade de escopos de quantificadores se restringiu a *Sistemas de Perguntas e Respostas*, onde geralmente as sentenças são autônomas a menos de elipses entre frases ou perguntas anafóricas do tipo *como*, *onde* e *porque*.

Embora todas as medidas são feitas para sentenças declarativas, nas sentenças interrogativas o material da sentença permanece inalterado havendo mudança somente de seu constituinte — como por exemplo, declarativo e interrogativo (verificativo ou identificativo).

Para tais sistemas, endereçamos a solução de que seria viável, através de um gerador de escopos, obter as várias interpretações do ponto de vista da ordem dos quantificadores. Sendo assim, a cada interpretação obtida do gerador, podemos, através de regras pragmáticas que raciocinem sobre quantidades e relações entre entidades, conseguir eliminar falsas interpretações para a sentença do usuário.

Através das medições realizadas e da comparação entre os dois algoritmos, obtivemos que o algoritmo de Hobbs e Shieber é muito mais lento que o de Pereira. Mas, como foi salientado no capítulo 7, a saída de um gerador deve ser acoplada aos Módulos Pragmático/Clarificação de Diálogos para que, *com certeza*, consigamos obter a interpretação correta da sentença.

Assim, as conclusões sobre qual algoritmo deve ser escolhido ficam prejudicadas se não avaliarmos todo o processo de conseguir a tradução correta.

Gostaríamos de deixar claro que em *Sistemas de PLN onde todo o diálogo é considerado*, a tarefa de um Módulo Pragmático se torna mais abrangente e talvez até mude a sua direção. Isto porque a tarefa da pragmática pode não ser a de selecionar um significado que interesse ao usuário dentre todos os possíveis significados que são obtidos da semântica.

Nestes casos, como mostra [Popov-86], a tarefa da pragmática seria dirigir — com ajuda de conhecimento do mundo e/ou contextual — o processo de interpretação, para se obter o significado semântico apropriado.

Fica, portanto, como sugestão para trabalhos futuros a continuidade do estudo de pragmática e a construção de um Módulo Pragmático/Clarificação de Diálogos em um sistema de Perguntas e Respostas. Desta forma, novas medidas podem

ser realizadas — referentes a todo o tempo gasto para encontrar a interpretação correta — para assim podermos tirar conclusões mais completas.

Bibliografia

- [Allen-80] Allen, J. F., Perrault, C. R. *Analysing Intention in Utterances*. Artificial Intelligence 15, pg. 143-178, 1980.
- [Allen-87] Allen, J. F. *Natural Language Understanding*. Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.
- [Arariboia-88] Arariboia, G. *Inteligência Artificial - Um Curso Prático*. LITECE, 1988.
- [Assis-86] Assis, F. S. G. de. *Uma Ferramenta de Auxílio no Desenvolvimento de Interfaces em Linguagem Natural*. Dissertação de Mestrado, IME 1986.
- [Barwise-87] Barwise, J. *Recent Developments in Situation Semantics*. Elsevier Science Publishers B.Y. (North-Holland), 1987.
- [Bierwisch-76] Bierwisch, M. *Novos Horizontes em Lingüística*. Cultrix, Universidade de São Paulo (eds), pg. 161-178, 1976.
- [Branco-87] Branco, A. C. S. *Semântica para Processamento de Linguagem Natural*. Dissertação de Mestrado, IME, Rio de Janeiro, 1987.
- [Carberry-88] Carberry, S. *Modeling the User's Plans and Goals*. Computational Linguistics, Vol. 14, Number 3, pg. 23-37, September 1988.
- [Carbonell-83] Carbonell, J. G.; Hayes, P. J. *A Tutorial on Techniques and Applications for Natural Language Processing*. Carnegie-Mellon University, 1983.
- [Clifford-88] Clifford, J. *Natural Language Querying of Historical Databases*. Computational Linguistics, Vol. 14, Number 4, pg. 10-34, December 1988.

- [Cohen-85] Cohen, R. *The Need for Pragmatics in Natural Language Understanding*. Proceedings of CSCSI 1985.
- [Colmerauer-77] Colmerauer, A. *An Interesting Subset of Natural Language*. G.I.A, Universite Aix-Marseille, France, pg. 45-66, October, 1977.
- [Dahl-81] Dahl, V. *Translating Spanish into Logic through Logic*. AJCL 7, 3, pg. 149-164, July-September, 1981.
- [Dascal-82] Dascal, M. *Fundamentos Metodológicos da Lingüística*. Dascal, M. (org). Campinas, Vol. 4 - Pragmática, IEL, pg. 7-22, 1982.
- [Grice-82] Grice, P. H. *Fundamentos Metodológicos da Lingüística*. Dascal, M. (org). Campinas, Vol. 4 - Pragmática, IEL, pg. 81-103, 1982.
- [Grosz-87] Grosz, B., Appelt, D., Martin, P. and Pereira, F. *TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces*. Artificial Intelligence 32, pg. 173-243, 1987.
- [Hadley-89] Hadley, R. F. *A Default-Oriented Theory of Procedural Semantics*. Cognitive Science 13, pg. 107-137, 1989.
- [Hess-85] Hess, M. - *About the Role of Control Information in Natural Language Question Answering Systems*. Natural Language Understanding and Logic Programming. V. Dahl and P. Saint-Dizier (ed), Elsevier Science Publishers B. V., North-Holland, pg. 165-180, 1985.
- [Hill-74] Hill, A. A. *Aspectos da Lingüística Moderna*. Cultrix, Universidade de São Paulo (eds), 1974.
- [Hirst-87] Hirst, G. *Encyclopedia of Artificial Intelligence*. Shapiro, S. C. (ed), New York: Wiley, pg. 1025-1029, 1987.
- [Hirst-88] Hirst, G. *Semantic Interpretation and Ambiguity*. Artificial Intelligence 34, pg. 131-177, 1988.
- [Hobbs-87] Hobbs, J. R. and Shieber, S. M. *An Algorithm for Generating Quantifier Scopings*. Computational Linguistics, Vol. 13, Numbers 1-2, pg. 47-63, January-June, 1987.
- [Kass-88] Kass, R., Finin T. *Modeling the User in Natural Language Systems*. Computacional Linguistics, Vol. 14, Number 3, pg. 5-51, September 1988.

- [Kempson-80] Kempson, R. M. *Teoria Semântica*. Zahar S.A. (ed), 1980.
- [Luft-85] Luft, C. P. *Moderna Gramática Brasileira*. Editora Globo S. A., 1985.
- [Maser-75] Maser, S. *Fundamentos de Teoria Geral da Comunicação*. EPU, EDUSP, 1975.
- [McCord-82] McCord, M. C. *Using Slots and Modifiers in Logic Grammars for Natural Language*. Artificial Intelligence 18, pg. 327-367, 1982.
- [McCord-87] McCord, M., Sowa, J. F., Wilson, W. G. and Walker, A. (eds). *A Logical Approach to Expert Systems and Natural Language Processing - Knowledge Systems and Prolog*. IBM T. J. Watson Research Center, Addison-Wesley Publishing Company, Inc., March, 1987.
- [McDermott-86] McDermott, D. *Tarskian Semantics, or No Notation Without Denotation* Readings in Natural Language Processing. Los Altos, CA, Morgan Kaufmann, pg. 167-169, 1986.
- [Melo-78] Melo, G.C. de. *Gramática Fundamental da Língua Portuguesa*. Rio de Janeiro: Ao Livro Técnico, 1978.
- [Miranda-86] Miranda, C. S. *Processamento de Linguagem Natural*. Dissertação de Mestrado, IME 1986.
- [Moore-86] Moore, R. C. *Problems in Logical Form*. Readings in Natural Language Processing. Los Altos, CA, Morgan Kaufmann, pg. 285-292, 1986.
- [Pereira-82] Pereira, F. C. N.; Warren, D. H. D. *An Efficient Easily Adaptable System for Interpreting Natural Language Queries*. AJCL 8, 3-4, pg. 110-122, July-December, 1982.
- [Pereira-83] Pereira, F. C. N. *Logic for Natural Language Analysis*. Technical Note número 275, 1983.
- [Pereira-86] Pereira, F. C. N.; Shieber, S. M. *Prolog and Natural-Language Analysis*. Curso Especial - IME, RJ, 1986.
- [Perrault-86] Cohen, P.R., Perrault, C. R. *Elements of a Plan-Based Theory of Speech Acts*. Readings in Natural Language Processing, Los Altos, CA, Morgan Kaufmann, pg. 423-440, 1986.

- [Popov-86] Popov, E.V. *Talking with Computers in Natural Language*. Springer-Verlag, 1986.
- [Rino-87] Rino, L. H. M. *Uma Interface em Linguagem Natural para Recuperação do Conhecimento*. Dissertação de Mestrado, ICMSC 1987.
- [Saint-Dizier-85] Saint-Dizier, P. *Handling Quantifier Scoping Ambiguities in a Semantic Representation of Natural Language Sentences*. Natural Language Understanding and Logic Programming. V. Dahl and P. Saint-Dizier (eds), Elsevier Science Publishers B. V., North-Holland, pg. 49-63, 1985.
- [Saint-Dizier-88] Saint-Dizier, P. *Default Logic, Natural Language and Generalized Quantifiers*. Proceedings of COLING, pg. 555-560, 1988.
- [Savadovsky-88] Savadovsky, P. *A Construção de Interpretadores para Linguagem Natural* - EBAI, 1988.
- [Schank-77] Schank, R. and Abelson, R. *Scripts, Plans, Goals and Understanding*. Hillsdale NJ: Lawrence Erlbaum, 1977.
- [Stalnaker-82] Stalnaker, R. C. *Fundamentos Metodológicos da Lingüística*. Dascal, M. (org). Campinas, Vol. 4 - Pragmática, IEL, pg. 59-80, 1982.
- [Sowa-84] Sowa, J. F. *Conceptual Structures - Information Processing in Mind and Machine*. Addison-Wesley Publishing Company, 1984.
- [Tondl-81] Tondl, L. *Problems of Semantics - A Contribution to the Analysis of Language of Science*. D. Reidel Publishing Company, Dordrecht, Holland, 1981.
- [Wilks-75] Wilks, Y. *A Preferential, Pattern-Seeking, Semantics for Natural Language Inference*. Artificial Intelligence 6, pg. 53-74, 1975.
- [Woods-86] Woods, W. A. *Semantics and Quantification in Natural Language Question Answering*. Readings in Natural Language Processing, Los Altos, CA, Morgan Kaufmann, pp. 205-248, 1986.
- [Woods-87] Woods, W. A. *Encyclopedia of Artificial Intelligence*. Shapiro, S. C. (ed), New York: Wiley, pg. 1029-1031, 1987.

Apêndice A

Listagens dos Programas Prolog

A.1 Interpretação Semântica e Gerador de Escopos de Pereira

```
:- op(910,xfy,&).  
:- op(920,xfy,=>).  
:- op(100,fx,...).  
:- op(50,fx,nao).
```

```
/* Programa para Apresentacao da Saida do Gerador de Escopos */
```

```
ini :- read_in(X), nl, mostre(X,[]).
```

```
mostre --> s(T), {write(T), nl, fail}.
```

```
mostre.
```

```
/* Programa para Teste de Tempo de Execucao */
```

```
mostre1(S) :- remover(S,T), fail.  
mostre1(S).
```

```
teststep1 :- read_in(X), s1(S,X,[]), tempo_exec(mostrel(S),100,TE).
```

```
/* Analise Sintatica e Interpretacao Semantica */
```

```
s(T) --> sint_nom(IV,S,G,N), sint_ver(IV,N), terminador,  
        {remove(S,T)}.
```

```
s1(S) --> sint_nom(IV,S,G,N), sint_ver(IV,N), terminador.
```

```
sint_nom(IV,S,G,N) --> det(N3,IV,S,G,N), subst_comum(N1,G,N),  
                        adjetivo(N1,N2,G,N), relativa(N2,N3,N).
```

```
sint_nom(IV,S,G,N) --> det(N3,IV,S,G,N), subst_rel(X^Y1),  
                        sint_prepos(Y1,X,N1), adjetivo(N1,N2,G,N),  
                        relativa(N2,N3,N).
```

```
sint_nom(E^S,S,G,s) --> subst_proprio(E,G).
```

```
sint_nom(V^S,S,G,N) --> cardinal(E,V,_), comum1(_,G,N).
```

```
sint_nom(E^S,S,G,N) --> artigo(G,N), subst_proprio(E,G,N,exige_artigo).
```

```
sint_ver(X^S,N) --> verbo(X^IV,N,td), sint_nom(IV,S,_,_).
```

```
sint_ver(X^S,N) --> verbo(X^IV,N,P), sint_prep(IV,S,P).
```

```
sint_ver(IV,N) --> verbo(IV,N,i).
```

```
sint_prep(IV,S,P) --> preposicao(P), sint_nom(IV,S,_,_).
```

```

sint_prepos(Y,X,X^N1) --> preposicao(P), sint_nom(Y,N1,_,_).

det(X^S1,X^S2,q(P^Q^todo(X,P=>Q),S1,S2),G,N) --> deter1(G,N).

det(X^S1,X^S2,q(P^Q^existe(X,P&Q),S1,S2),G,N) --> deter2(G,N).

det(X^S1,X^S2,q(P^Q^ nao(existe(X,P&Q)),S1,S2),G,N) --> deter3(G,N).

relativa(X^S1,X^ (S1&S2),N) --> pronome_relativo, sint_ver(X^S2,N),!.

relativa(X^S1,X^ (S1&S2),N) --> preposicao(P), pronome_relativo,
                                sint_ver(X^S2,N),!.

relativa(N,N,_) --> [].

/* Simbolos Terminais */

terminador --> ([.];[?],['!']).

artigo(m,s) --> [o], !.
artigo(f,s) --> [a], !.
artigo(m,p) --> [os], !.
artigo(f,p) --> [as], !.

deter1(f,s) --> [toda], !.
deter1(_,s) --> ([cada];[qualquer]), !.
deter1(f,p) --> [todas, as], !.
deter1(_,p) --> [quaisquer], !.
deter1(m,s) --> [todo], !.
deter1(m,p) --> [todos, os], !.

deter2(f,s) --> ([uma];[alguma];[a]), !.
deter2(f,p) --> ([umas];[algumas]), !.
deter2(m,s) --> ([um];[algum];[o]), !.
deter2(m,p) --> ([uns];[alguns]), !.

```



```

deter2(f,p) --> [as], !.
deter2(m,p) --> [os], !.

deter3(f,s) --> [nenhuma], !.
deter3(m,s) --> [nenhum], !.

preposicao(P) --> [P],{prep(P)},!.
preposicao(P) --> [C],{contr(C,P,Art)},!.

pronome_relativo --> [que],!.
pronome_relativo --> [quem],!.
pronome_relativo --> [o,qual],!.
pronome_relativo --> [a,qual],!.

verbo(X^IV,N,P) --> [Verbo], {v(Verbo,X^IV,N,P)}.

adjetivo(X^S1,X^ (S1&S2),G,N) --> [Adjetivo],
                                   {adj(Adjetivo,X^S2,G,N)}, !.

adjetivo(N,N,_,_) --> [].

subst_comum(X^ P,G,N) --> [Nome], {n(Nome,X^P,G,N)}.

subst_proprio(Nome,G) --> [Nome], {np(Nome,G)}.
subst_proprio(Nome,G,N,exige_artigo) --> [Nome],
                                           {np(Nome,art,G,N)}.

cardinal(Numero,Num,_) --> [Numero], {car(Numero,Num,_)}.

comum1(Nome,G,N) --> [Nome], {n1(Nome,G,N)}.

subst_rel(S) --> [NomeRel], {nr(NomeRel,S)}.

sint_adj(IV,G,N) --> [Adjetivo], {adj(Adjetivo,IV,G,N)}.

/* Pequeno Dicionario */

```

/* Genero e Numero de Nomes Comuns e Adjetivos */

n(estudantes,X^ (...estudante(X)),_,p).

n(microcomputador,X^ (...microcomputador(X)),m,s).

n(vax,X^ (...vax(X)),m,_).

n(rio,X^ (...rio(X)),m,s).

n(pais,X^ (...pais(X)),m,s).

n(guerreiro,X^ (...guerreiro(X)),m,s).

n(companhia,X^ (...companhia(X)),f,s).

n(amostra,X^ (...amostra(X)),f,s).

n(picos,X^ (...pico(X)),m,p).

n(tropas,X^ (...tropa(X)),f,p).

n(papa,X^ (...papa(X)),m,s).

n(guerreiros,X^ (...guerreiros(X)),m,p).

n(lanca,X^ (...lanca(X)),f,s).

n(banco,X^ (...banco(X)),m,s).

n(navio,X^ (...navio(X)),m,s).

n(cachorro,X^ (...cachorro(X)),m,s).

n(gato,X^ (...gato(X)),m,s).

n(bandido,X^ (...bandido(X)),m,s).

n(departamento,X^ (...departamento(X)),m,s).

n(produto,X^ (...produto(X)),m,s).

```

n(esposa,X^ (...esposa(X)),f,s).

n(regresso,X^ (...regresso(X)),m,s).

n(terra,X^ (...terra(X)),f,s).


adj(europeu,X^ (...europeu(X)),m,s).

adj(europeus,X^ (...europeu(X)),m,p).

adj('sul americano',X^ (...'sul americano'(X)),m,s).

adj(gratis,X^ (...gratis(X)),f,_).

adj(bonita,X^ (...bonita(X)),f,s).

adj(natal,X^ (...natal(X)),f,s).


/* Genero e Numero dos Nomes Proprios */

np('revolucao francesa',art,f,s).

np(brasil, _).
np('sao carlos', _).

car(mil,1000,_).

n1(metros,_,p).


/* Preposicoes e Contracoes*/

prep(de).
prep(em).
prep(com).
prep(a).
prep(para).

contr(do,de,o).

```

contr(da,de,a).
contr(no,em,o).
contr(na,em,a).

/* Nomes que expressam Relacoes */

nr(presidente,X^Y^ (...presidente(X,Y))).

nr(vigia,X^Y^ (...vigia(X,Y))).

nr(representante,X^Y^ (...representante(X,Y))).

nr(altura,X^Y^ (...altura(X,Y))).

nr(comandante,X^Y^ (...comandante(X,Y))).

nr(policial,X^Y^ (...policial(X,Y))).

nr(capital,X^Y^ (...capital(X,Y))).

nr(processos,X^Y^ (...processo(X,Y))).

/* Verbos*/

v(usam,X^Y^ (...usar(X,Y)),p,td).

v(atravessa,X^Y^ (...atravessar(X,Y)),s,td).

v(visitou,X^Y^ (...visitar(X,Y)),s,td).

v(trouxe,X^Y^ (...trazer(X,Y)),s,td).

v(ultrapassa,X^Y^ (...ultrapassar(X,Y)),s,td).

v(tem,X^Y^ (...ter(X,Y)),p,td).

v(embarcaram,X^Y^ (...embarcar(X,Y)),p,em).

v(embarcou,X^Y^ (...embarcar(X,Y)),s,em).

v(persegue,X^Y^ (...perseguir(X,Y)),s,td).

```

v(assaltou,X^Y^ (...assaltar(X,Y)),s,td).

v(festejou,X^Y^ (...festejar(X,Y)),s,td).

v(analisou,X^Y^ (...analisar(X,Y)),s,td).

v(espera,X^Y^ (...espera(X,Y)),s,td).

v(voltou,X^Y^ (...voltar(X,Y)),s,para).


/* Geracao de Escopos Alternativos */

remove(QuantTree,Formula) :- remove(QuantTree,Matrix,Store),
    aplicar_quants(Store,Matrix,Formula).

remove(...Predication,Predication,[]).

remove(QuantTree1 & QuantTree2, Formula1 & Formula2,Store) :-
    remove(QuantTree1,Matrix1,Store1),
    remove(QuantTree2,Matrix2,Store2),
    concatenar(Pass1,Apply1,Store1),
    concatenar(Pass2,Apply2,Store2),
    aplicar_quants(Apply1,Matrix1,Formula1),
    aplicar_quants(Apply2,Matrix2,Formula2),
    misturar(Pass1,Pass2,Store).

remove(q(Range^Quant, RangeTree, ScopeTree),Matrix, Store) :-
    remove(RangeTree,RangeMatrix,RangeStore),
    remove(ScopeTree,Matrix,ScopeStore),
    concatenar(RangePass,RangeApply,RangeStore),
    aplicar_quants(RangeApply,RangeMatrix,Range),
    concatenar(RangePass,[Quant],Pass),
    misturar(Pass,ScopeStore,Store).

misturar([],[],[]).

misturar([Elem|LeftList],RightList,[Elem|List]) :-

```

```

misturar(LeftList,RightList,List).

misturar(LeftList,[Elem|RightList],[Elem|List]) :-
    misturar(LeftList,RightList,List).

aplicar_quants([],Formula,Formula).

aplicar_quants([SubFormula^Formula|Quants],Matrix,Formula) :-
    aplicar_quants(Quants,Matrix,SubFormula).

concatenar([],List,List).

concatenar([Elem|Front],Back,[Elem|List]) :-
    concatenar(Front,Back,List).

```

A.2 Gerador de Escopos de Hobbs e Shieber

% Definicao de alguns determinadores e operadores opacos

```

quantificador(algum).
quantificador(alguma).
quantificador(alguns).
quantificador(algumas).
quantificador(todo).
quantificador(toda).
quantificador(um).
quantificador(uma).
quantificador(o).
quantificador(a).
quantificador(qualsquer).
quantificador(nenhuma).
quantificador(nenhum).
quantificador(qualquer).
quantificador(cada).
quantificador(onde).
quantificador(uns).
quantificador(umas).

```

```
quantificador('todos os').
quantificador('todas as').
quantificador(os).
quantificador(as).
```

```
opaco(nao,1).
```

```
/* Programa para Impressao da Saida do Gerador de Escopos */
```

```
hob(ID) :- tes(ID,W), mostra(W).
```

```
mostra(W) :- gerar(W,S), write(S), nl, fail.
mostra(W).
```

```
/* Programa para Teste de Tempo de Execucao */
```

```
testeh1(ID) :- tes(ID,W), tempo_exec(mostrat(W),100,TE).
```

```
mostrat(W) :- gerar(W,S), fail.
mostrat(W).
```

```
/* Exemplos de Linguagem de Entrada */
```

```
tes(1,fbf(usar,[termo('todos os',e,fbf(estudante,[e])),
                termo(um,m,fbf(microcomputador,[m]))])).
```

```
tes(2,fbf(usar,[termo('todos os',e,fbf(estudante,[e])),
                termo(um,v,fbf(vax,[v]))])).
```

```
tes(3,fbf(atravessar,[termo(um,r,fbf(rio,[r])),
                termo(cada,p,fbf(and,[fbf(pais,[p]),fbf(europeu,[p])]))])).
```

```
tes(4,fbf(visitar,[termo(cada,p,fbf(and,
                                [fbf(presidente,[p]),
                                fbf(de,[p,termo(um,pa,fbf(and,
                                [fbf(pais,[pa,p]),
                                fbf('sul americano',[pa]))]))]),
                                brasilia]))).
```

```
tes(5,fbf(festear,[termo(a,c,fbf(and,
                                [fbf(capital,[c]),
                                fbf(de,[c,termo(cada,pa,fbf(and,
                                [fbf(pais,[pa,c]),
                                fbf(europeu,[pa]))]))]),
                                'a Revolucao Francesa'))).
```

```
tes(6,fbf(ultrapassar,[termo(a,al,fbf(and,
                                [fbf(altura,[al]),
                                fbf(de,[al,termo('todos os',p,fbf(and,
                                [fbf(picos,[p,al]),
                                fbf(europeus,[p]))]))]),
                                'mil metros'))).
```

```
tes(7,fbf(ter,[termo(os,g,fbf(guerreiros,[g])),
                termo(uma,l,fbf(lanca,[l])))]).
```

```
tes(8,fbf(embarcar,[termo(os,g,fbf(guerreiros,[g])),
                termo(um,n,fbf(navio,[n])))]).
```

```
tes(9,fbf(perseguir,[termo(todo,c,fbf(cachorro,[c])),
                termo(um,g,fbf(gato,[g])))]).
```

```
tes(10,fbf(dorme,[termo(todo,v,fbf(and,[fbf(vigia,[v]),
                fbf(de,[r,termo(uma,c,fbf(companhia,[c,v]))]))])),
                ]).
```



```
tes(11,fbf(trazer,[termo(cada,r,fbf(and,
                                [fbf(representante,[r]),
                                fbf(de,[r,termo(uma,c,
                                fbf(companhia,[c,r]))]))]),
                                termo(uma,a,fbf(and,[fbf(amostra,[a]),fbf(gratis,[a]))])))).
```

```
tes(12,fbf(visitar,[termo(o,c,fbf(and,
                                [fbf(comandante,[c]),
                                fbf(de,[c,termo('todas as',t,
                                fbf(tropas,[t,c]))]))]),
                                termo(o,p,fbf(papa,[p]))])).
```

```
tes(13,fbf(perseguir,[termo(cada,p,fbf(and,
                                [fbf(policial,[p]),
                                fbf(de,[p,'sao carlos'])])),
                                termo(o,b,fbf(and,[fbf(bandido,[b]),
                                fbf(assaltar,
                                [termo(um,ba,fbf(banco,[ba]))]))])))).
```

```
tes(14,fbf(voltar,[termo(nenhum,c,fbf(and,
                                [fbf(guerreiro,[g]),
                                fbf(embarcar,[termo(um,na,
                                fbf(navio,[na]))]))]),
                                fbf(and,[termo(a,t,fbf(terra,[t]),
                                fbf(natal,[t]))])))).
```

```
tes(15,fbf(analisar,[termo(um,r,fbf(and,
                                [fbf(representante,[r]),
                                fbf(de,[r,termo(cada,d,
                                fbf(departamento,[d,r]))]))]),
                                termo('todos os',p,fbf(and,[fbf(processos,[p]),
                                fbf(de,[p,termo(cada,pr,fbf(produto,[pr,p]))]))])))).
```

```
tes(16,fbf(ter,[termo(os,g,fbf(and,
                                [fbf(guerreiros,[g]),
                                fbf(embarcar,[termo(um,n,fbf(navio,[n]))]))]),
```

```

        termo(uma,e,fbf(and,[fbf(esposa,[e]),fbf(bonita,[e]),
            fbf(esperar,
                [termo(o,re,fbf(regresso,[re]))])))))).

```

```

/* Gerador de Escopos de Hobbs e Shieber */

```

```

% gerar(Form,ScopedForm)

```

```

%

```

```

gerar(Form,ScopedForm) :-

```

```

    remover(Form,true,ScopedForm).

```

```

% remover(Form,Completo?,ScopedForm)

```

```

%

```

```

remover(Form,Complete,ScopedForm) :-

```

```

    remover_args_opacos(Form,PulledOpaque),

```

```

    aplicar_termos(PulledOpaque,Complete,ScopedForm).

```

```

% remover_args_opacos(Form,ScopedForm)

```

```

%

```

```

remover_args_opacos(fbf(Pred,Args),fbf(Pred,ScopedArgs)) :- !,

```

```

    remover_args_opacos(Pred, 1, Args,ScopedArgs).

```

```

remover_args_opacos(Term,Term).

```

```

% remover_args_opacos(Pred,ArgIndex,Args,ScopedArgs)

```

```

% Fim dos argumentos

```

```

remover_args_opacos(_, _, [],[]) :- !.

```

```

% Argumento corrente e' opaco

```

```

remover_args_opacos(Pred,ArgIndex,

```

```

    [FirstArg|RestArgs],[ScopedFirstArg|ScopedRestArgs]) :-

```

```

    opaco(Pred,ArgIndex), !,

```

```

    remover(FirstArg,false,ScopedFirstArg),

```

```

    NextIndex is ArgIndex+1,

```

```

    remover_args_opacos(Pred,NextIndex,RestArgs,ScopedRestArgs).

% Argumento corrente nao e' opaco
remover_args_opacos(Pred,ArgIndex,
    [FirstArg|RestArgs],[ScopedFirstArg|ScopedRestArgs]) :-
    remover_args_opacos(FirstArg,ScopedFirstArg),
    NextIndex is ArgIndex+1,
    remover_args_opacos(Pred,NextIndex,RestArgs,ScopedRestArgs).

% aplicar_termos(Form,Complete?,ScopedForm)
%
aplicar_termos(Form,_,Form) :-
    not(termo(Form,_)), !.

aplicar_termos(Form,false,Form).

aplicar_termos(Form,Complete,ScopedForm) :-
    termo_aplicavel(Form,Term),
    aplicar(Term,Form,AppliedForm),
    aplicar_termos(AppliedForm,Complete,ScopedForm).

% aplicar(Term,Form,NewForm)
%
aplicar(termo(Quant,Var,Restrict),Body,
    fbf(Quant,[Var,PulledRestrict,OutBody])) :-
    remover(Restrict,false,PulledRestrict),
    subst(Var,termo(Quant,Var,Restrict),Body,OutBody).

% termo_aplicavel(Form,Term)
%
termo_aplicavel(Form,Term) :-
    termo_aplicavel(Form,Term,[]).

% termo_aplicavel(Form,Term,BlockingVars)
%

% Um termo e um termo aplicavel do topo

termo_aplicavel(termo(Q,V,R),termo(Q,V,R),BVs) :-
    not(livre(BVs,R)).

```

```

termo_aplicavel(fbf(Quant,[Var,Restrict,Body]),Term,BVs) :-
    quantificador(Quant), !,
    (termo_aplicavel(Restrict,Term,[Var|BVs]);
     termo_aplicavel(Body,Term,[Var|BVs])).

```

```

termo_aplicavel(fbf(_,Args),Term,BVs) :-
    termo_aplicavel(Args,Term,BVs).

```

```

termo_aplicavel([F|R],Term,BVs) :-
    (termo_aplicavel(F,Term,BVs) ;
     termo_aplicavel(R,Term,BVs)).

```

% Rotinas Utilitarias para a Geracao de Escopos

```

termo(termo(Q,V,R),termo(Q,V,R)).
termo(fbf(_,Args),Term) :- termo(Args,Term).
termo([F|R],Res) :- (termo(F,Res) ; termo(R,Res)).

```

```

livre([Var|Vars],Form) :- (rec_membro(Var,Form) ; livre(Vars,Form)).

```

% Utilidades Genericas

```

rec_membro(Element,Element) :- !.
rec_membro(_,Other) :- atomic(Other), !, fail.
rec_membro(Element,[First|Rest]) :- !,
    (rec_membro(Element,First) ; rec_membro(Element,Rest)).
rec_membro(Element,Term) :- Term =.. List, rec_membro(Element,List).

```

```

subst(New,Old,Old,New) :- !.
subst(_,_,[],[]) :- !.
subst(_,_,Atom,Atom) :- atomic(Atom), !.
subst(New,Old,[First|Rest],[OutFirst|OutRest]) :- !,
    subst(New,Old,First,OutFirst),
    subst(New,Old,Rest,OutRest).
subst(New,Old,In,Out) :-
    In =.. List, subst(New,Old,List,OutList),
    Out =.. OutList.

```

```

not(F) :- F, !, fail.

```

not(_).

retirar(_,[],[]).

retirar(Element,[Element|List],NewList) :- !,
 retirar(Element,List,NewList).

retirar(Element,[Other|List],[Other|NewList]) :-
 retirar(Element,List,NewList).

A.3 Programa para Medir o Tempo de Execucao

```
/*
tempo_exec(+Pred, +Nvezes, ?Tempo)

Executa o predicado Pred Nvezes dando o tempo de execucao Tempo
ja retirado o tempo do loop repeat
*/
tempo_exec(Pred,Nvezes,Tempo) :-
time(T1), /* tempo inicial */
execute(Pred,Nvezes),
time(T2), /* tempo final total */
loop_repeat(Nvezes), /* executa o loop repeat */
time(T3),
tempo(T2-T1,T12), /* T12 tempo total */
tempo(T3-T2,T23), /* T23 tempo usado pelo loop_repeat */
Tempo is T12 - T23, /* tempo real usado na execucao de Pred */
escreva1(Pred,Nvezes,Tempo),
!.

/*
execute(+Pred, +Nvezes)

Executa o predicado Pred Nvezes dentro de um repeat loop
*/
execute(Pred,Nvezes) :-
ctr_set(0,1),
```

```

repeat,
Pred,
ctr_inc(0,X),
X == Nvezes.

/*
loop_repeat(+Nvezes)

Executa um "dummy" loop_repead Nvezes
*/
loop_repeat(Nvezes) :-
ctr_set(0,1),
repeat,
ctr_inc(0,X),
X == Nvezes.

/*
tempo(+F-I, ?T)
calcula T = F - I onde F e I sao time(H,M,S,Cs)
*/

tempo(time(Hf,Mf,Sf,Csf) - time(Hi,Mi,Si,Csi), T) :-
H is (Hf - Hi) * 360000,
M is (Mf - Mi) * 6000,
S is (Sf - Si) * 100,
Cs is Csf - Csi,
T is H + M + S + Cs.

escreva1(Pred,Nvezes,Tempo) :-
nl,
tab(10),write('Predicado = '),write(Pred),nl,
tab(10),write('No_vezes = '),write(Nvezes),nl,
tab(10),write('Tempo = '),write(Tempo),nl,
Tmedio is Tempo / Nvezes,
tab(10),write('Tmedio = '),write(Tmedio),nl.

```